# Asynchronous Verifiable Information Dispersal

Christian Cachin Stefano Tessaro

IBM Research
Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{cca,tes}@zurich.ibm.com

November 15, 2004

**Abstract**

*Information dispersal* addresses the question of storing a file by distributing it among a set of servers in a storage-efficient way. We introduce the problem of *verifiable* information dispersal in an *asynchronous* network, where up to one third of the servers as well as an arbitrary number of clients might exhibit Byzantine faults. Verifiability ensures that the stored information is consistent despite such faults. We present a storage- and communication-efficient scheme for asynchronous verifiable information dispersal that achieves an asymptotically optimal storage blow-up. Additionally, we show how to guarantee the secrecy of the stored data with respect to an adversary that may mount adaptive attacks. Our technique also yields a new protocol for asynchronous reliable broadcast that improves the communication complexity by an order of magnitude on large inputs.

## 1 Introduction

With the increasing availability of fast networks, distributed storage systems have become an attractive solution for managing large amounts of information. New technologies such as NAS and SAN connect storage devices directly to clients. This development also introduces new security and dependability problems because the devices may exhibit failures or become a target for attacks, which are not linked to the clients.

With this background, we consider following problem: Some data has to be distributed by a *client* among a set of $n$ *servers*, of which up to $t$ might be faulty exhibiting *Byzantine* behavior (that is, they may deviate arbitrarily from the protocol), in such a way that clients can always recover the stored data correctly, independently from the behavior of the faulty servers. One trivial but inefficient solution is to use *replication* such that every server keeps a copy of the data. The classic alternative, as proposed by Rabin [Rab89], is an *information dispersal algorithm* (IDA): using an *erasure code*, the data is split into blocks, so that each server holds exactly one block, and so that only a subset of the blocks is needed in order to reconstruct the data.

A protocol for information dispersal based on IDA tolerating Byzantine servers has been proposed by Garay et al. [GGJR00]. It relies on *synchronous* networks, which are adequate for tightly coupled nodes such as clusters, but unrealistic for geographically distributed or heterogeneous systems. An *asynchronous* network model is more appropriate for such settings.

One issue that has not received much attention so far is the presence of *Byzantine clients*. A first solution for erasure-coded storage in an asynchronous network that prevents some problems caused by faulty clients has recently been proposed by Goodson et al. [GWGR04]. However, it relies on correct clients to establish a consistent state across the servers. We consider the consistency of the service state to be a desirable goal in order to protect innocent clients from obtaining inconsistent values, although one may also argue, from a client-oriented perspective, that problems caused by faulty clients such as inconsistently stored data do not hurt the system.

In this paper, we introduce the notion of *verifiability* for information dispersal in *asynchronous* networks with a computationally bounded adversary. Intuitively, verifiability means that whenever the honest servers accept to store some data, then the data is also consistent and no two distinct honest clients can reconstruct different data. This notion of verifiability originates in the related context of secret sharing, where various protocols for *verifiable secret sharing* exist, both in synchronous [Fel87, Ped92] and asynchronous networks [CKLS02].

We propose a new scheme for asynchronous verifiable information dispersal that is also storage- and communication-efficient and achieves optimal resilience $t < \frac{n}{3}$. It combines an asynchronous reliable broadcast protocol with erasure coding and achieves a communication complexity of $\mathcal{O}(n|F|)$ bits for storing a file $F$, which is an order of magnitude more efficient than what results from the previously known approach, which needs $\mathcal{O}(n^2|F|)$ bits. The storage blow-up of the scheme, i.e., the storage space needed in relation to $|F|$, is asymptotically optimal.

Several optimizations of the scheme are presented, including one that uses error-*correcting* codes (ECC) for lowering the storage requirements; one of the optimized schemes leads directly to a communication-efficient protocol for *asynchronous reliable broadcast* with Byzantine faults. For broadcasting a large message $m$, this protocol communicates only $\mathcal{O}(n|m|)$ bits instead of $\mathcal{O}(n^2|m|)$.

We also consider how to provide *confidentiality* for the stored data. Clients might want to store data that should only be read by authorized users. We present an extension of our asynchronous verifiable information dispersal scheme that incorporates a threshold cryptosystem for secrecy, but maintains the communication and storage efficiencies of the scheme without confidentiality.

## 1.1 Related work

Erasure codes are well known in coding theory [Bla83]. Rabin's work [Rab89] introduces the concept of information dispersal algorithms (IDA) for splitting large files, but does not address protocol aspects for implementing IDA in distributed systems.

IDA is extended by Krawczyk [Kra93] using a technique called *distributed fingerprinting* in order to ensure the integrity of data in case of alterations of the stored blocks by malicious servers. The same idea is subsequently improved by Alon et al. [AKK$^+$00, AKK$^+$04].

Garay et al. [GGJR00] propose an information dispersal scheme for synchronous networks. Their model does not allow Byzantine clients, even though some attacks are tolerated. A key concept is the so-called *gateway*, an honest party through which clients access the servers comprising the storage system. The gateway is actually implemented by one of the servers and might be corrupted, but additional measures prevent it from causing too much damage: encryption hides the data from the gateway and a time-out mechanism rotates the gateway function to another server, should the first server not respond properly and in time. Clients only communicate with the gateway. Because of its inherent synchrony, this protocol cannot be translated to an asynchronous network.

The dispersal protocol of Garay et al. consists of two steps: first, the file is broadcast to all servers; then every server applies a (deterministic) information dispersal algorithm, which yields $n$ blocks, one for each server. Every server stores its own block and erases all others. An asynchronous dispersal protocol without secrecy is easily obtained from this by eliminating the gateway and implementing the broadcast using an asynchronous reliable broadcast protocol that tolerates Byzantine faults, such as Bracha's protocol [Bra84]. We discuss this protocol in Section 3.2 and refer to it as *asynchronous GGJR* below; it satisfies our verifiability property, but its communication blow-up is $\mathcal{O}(n^2)$.

A solution for erasure-coded storage in an asynchronous network with robustness against Byzantine clients has recently been proposed by Goodson et al. [GWGR04]. However, inconsistently written data can only be detected at read-time; the content of the storage system is then rolled back to restore the data to the last correctly written data. The major drawback of this approach is that retrieving data can

| Scheme | Model | Storage Blow-up | Communication Blow-up |
|---|---|---|---|
| [GWGR04] | $t < \frac{n}{4}$, no verifiability | $\frac{n}{n-3t} + o(1)$ | $\mathcal{O}(1)$ |
| asynchronous GGJR | $t < \frac{n}{3}$, verifiability | $\frac{n}{n-t} + o(1)$ | $\mathcal{O}(n^2)$ |
| AVID (this paper) | $t < \frac{n}{3}$, verifiability | $\frac{n}{n-2t} + o(1)$ | $\mathcal{O}(n)$ |
| AVID-RBC (this paper) | $t < \frac{n}{3}$, verifiability | $\frac{n}{n-t} + o(1)$ | $\mathcal{O}(n)$ |
| AVID-ECC (this paper) | $t < \frac{n}{4}$, verifiability | $\frac{n}{n-3t}$ | $\mathcal{O}(n)$ |

Table 1: Comparison of asynchronous information dispersal schemes for large files. Communication blow-up refers to the dispersal protocol.

be very inefficient in the case of several failed write operations, and that consistency depends on correct clients. The protocol requires $t < \frac{n}{4}$.

Table 1 summarizes the performance of the schemes discussed so far in terms of their storage blow-up and the communication blow-up of the dispersal protocol. They are compared with the main scheme of the paper (AVID from Section 3.3), with the optimization using reliable broadcast (AVID-RBC, Section 3.7), and with the variation using ECC (AVID-ECC, Section 3.7). In all schemes the communication blow-up of the retrieval protocol is $\mathcal{O}(1)$ (for the scheme of Goodson et al. [GWGR04] this holds only if no inconsistent data has been written).

Our main scheme AVID achieves a better storage blow-up and tolerates more corrupted servers than the protocol of Goodson et al. [GWGR04], which has a better (i.e., constant) communication blow-up, but does not provide verifiability. Obtaining verifiability seems to cause an increase by a factor of $\Theta(n)$ in the communication complexity. Intuitively, it seems plausible that every honest server has to see the whole file in order to decide whether its block is consistent with the rest or not, but we are not aware of any formal lower bounds.

### 1.2 Outline of the paper

In Section 2 we introduce our asynchronous network model with Byzantine corruptions and give a definition for *asynchronous verifiable information dispersal*. Section 3 is devoted to presenting our asynchronous verifiable information dispersal scheme called AVID. Several optimizations are also presented in this section, and it is shown how to implement asynchronous *reliable broadcast* from verifiable information dispersal. The verifiable information dispersal scheme with confidentiality, called cAVID, is described in Section 4.

## 2 Definitions

In this section, we first introduce our system model for describing asynchronous protocols. We then give a definition for asynchronous verifiable information dispersal and a refined definition for information dispersal with confidentiality. The complexity measures used in the analysis of our protocols and a few basic tools are also introduced.

### 2.1 System model

We use a model which is similar to the one in [CKPS01], even though we adopt some simplifications.

The network consists of a set of *servers* $\{P_1, \ldots, P_n\}$ and a set of *clients* $\{C_1, C_2, \ldots\}$, which are all probabilistic interactive Turing machines (PITM) with running time bounded by a polynomial in a given security parameter $\kappa$. Servers and clients together are called *parties*. There is an *adversary*, which is a PITM with running time bounded by a polynomial in $\kappa$. Servers and clients can be controlled by the adversary. In this case, they are called *corrupted*, otherwise they are called *honest*. An adversary that controls up to $t$ servers is called $t$*-limited*. We are not assuming any bounds on the number of clients that can be corrupted. The adversary is *static*, that is, it must choose the parties it corrupts before starting the protocol. Additionally, there is an initialization algorithm, which is run by some trusted party before the system actually starts.

Every pair of servers is linked by a *secure asynchronous channel* that provides privacy and authenticity with scheduling determined by the adversary. In contrast to [CKPS01], we consider only a benign adversary that may delay messages but will eventually deliver every message between honest parties. Moreover, every client and every server are linked by a secure asynchronous channel.

Whenever the adversary delivers a message to an honest party, this party is *activated*. In this case, the message is put in a so called input buffer, the party reads then the content of its buffer, performs some computation, and generates one or more response messages, which are written on the output tape of the party.

Protocols can be invoked either by the adversary, or by other protocols. Every protocol instance is identified by a unique string *ID*, called the *tag*, which is chosen arbitrarily by the adversary if it invokes the protocol, or which contains the tag of the calling protocol as a prefix if the protocol has been invoked by some other protocol. There may be several threads of execution for a given party, but only one of them is allowed to be active concurrently. When a party is activated, all threads are in *wait states*, which specify a condition defined on the received messages contained in the input buffer, as well as on some local variables. If one or more threads are in a wait state whose condition is satisfied, one of these threads is scheduled (arbitrarily) and this thread runs until it reaches another wait state. This process continues until no more threads are in a wait state whose condition is satisfied. Then, the activation of the party is terminated and the control returns to the adversary.

We distinguish between *local events*, which are either *input actions* (that is, messages of the form $(\mathit{ID}, \mathtt{in}, \mathit{type}, \ldots)$) or *output actions* (messages of the form $(\mathit{ID}, \mathtt{out}, \mathit{type}, \ldots)$), and other *protocol messages*, which are ordinary protocol messages to be delivered to other parties (of the form $(\mathit{ID}, \mathit{type}, \ldots)$). All messages of this form that are generated by honest parties are said to be *associated* to the protocol instance *ID*.

Whenever an output action is performed for one party, then the protocol terminates for this party, that is all the threads of the same instance are stopped.

We use the following syntax for specifying our protocols. To enter a wait state, a thread executes a command of the form **wait for** *condition*. There is a global implicit **wait for** statement that every protocol instance repeatedly executes: it matches any of the *conditions* given in the clauses of the form **upon** *condition block*.

## 2.2 Asynchronous verifiable information dispersal

The basic data item one client wants to store in a storage system is a *file*. An *asynchronous verifiable information dispersal (AVID) scheme* for a file $F$ consists of two protocols:

**The dispersal protocol:** A client starts this protocol as it decides to store a certain file $F$ in the storage system provided by the $n$ servers. Some redundancy is added to the file, which is then split into $n$ different blocks, each one being stored by one of the $n$ servers.

**The retrieval protocol:** A client (not necessarily the same which has written the file $F$), wanting to retrieve file $F$, invokes this protocol in order to receive enough information from the servers to reconstruct the file $F$. Moreover, the retrieval protocol can be repeated as many times as necessary.

Note that we do not address the questions of concurrency and versioning. A file $F$ can be written only once, but retrieved again and again. Since updates are not possible, concurrent reads and writes are not a problem. Stored files are indexed using the tag *ID* of the instance of the dispersal protocol which wrote them. Therefore, running the retrieval protocol for *ID* simply means retrieving the file stored with the instance of the dispersal protocol with tag *ID*.

We say that a client *disperses* a file $F$ for *ID* if it starts the dispersal protocol with tag *ID* with a file $F$ as an input, that is, it is activated through an input action $(\mathit{ID}, \texttt{in}, \texttt{disperse}, F)$. Furthermore, a server may *complete* the dispersal *ID* if it terminates the dispersal protocol for *ID* with some output of type `stored`, and it may *abort* the dispersal *ID* if it terminates the protocol with an output of type `abort`. However, a server might neither complete nor abort the dispersal. Finally, a client *reconstructs* a file $F'$ for *ID*$'$ if it terminates the retrieval protocol for the file stored with tag *ID*$'$ with an output $(\mathit{ID}, \texttt{out}, \texttt{retrieved}, F')$.

The verifiability property requires that either all servers complete the dispersal or no server completes the dispersal. This ensures that the servers always store consistent data once enough honest servers have accepted. This is formalized in the following definition.

**Definition 1.** A $(k, n)$-*asynchronous verifiable information dispersal scheme* ($k \leq n$) is composed by a dispersal and a retrieval protocol which satisfy, for any $t$-limited adversary, any *ID*, and any client $C_i$ starting the dispersal protocol for *ID*, the following conditions, except with negligible probability:

**Termination:** If the client $C_i$ is honest, then all honest servers eventually complete the dispersal *ID*.

**Agreement:** If some honest server completes the dispersal *ID*, then all honest servers eventually complete the dispersal *ID*.

**Availability:** If $k$ honest servers complete the dispersal *ID*, and an honest client $C_j$ starts the retrieval protocol for *ID*, then it eventually reconstructs some file $F'$.

**Correctness:** If $k$ honest servers complete the dispersal *ID*, there exists a fixed value $G$ such that the following holds:

1. If $C_i$ is honest and has dispersed a file $F$ using *ID*, then $G = F$.
2. If an honest client $C_j$ reconstructs $F'$ for *ID*, then $G = F'$.

In order use information dispersal in applications where information must be kept secret, a scheme must satisfy stronger requirements. First of all, one has to define who may retrieve a stored file and who may not, since we want that a file can be retrieved not only by the client who stored it. To this end, every file $F$ is stored with an associated *access list* $\mathcal{L}$, consisting of the set of indices of the clients allowed to retrieve $F$. We require that the adversary does not gain knowledge about the stored file unless a corrupted client is in the access list of the file. The adversary may interact with the servers in arbitrary ways apart from that, which allows it to mount the equivalent of an adaptive chosen-ciphertext attack. This is captured by the following definition.

**Definition 2.** A $(k, n)$-*asynchronous verifiable information dispersal scheme with confidentiality* ($k \leq n$) is composed by a dispersal and a retrieval protocol which satisfy, for any $t$-limited adversary, any *ID*, and any client $C_i$ starting the dispersal protocol, the Termination, Agreement, and Availability properties of Definition 1, as well as the following properties:

**Correctness:** If $k$ honest servers complete the dispersal *ID*, there exists a fixed value $G$ and a fixed access list $\mathcal{M}$ such that the following holds, except with negligible probability:

1. If $C_i$ has dispersed a file $F$ with access list $\mathcal{L}$ using *ID* and is honest, then $F = G$ and $\mathcal{L} = \mathcal{M}$.
2. If $C_j, j \in \mathcal{M}$, is honest and reconstructs $F'$ for *ID*, then $F' = G$.
3. If $C_j, j \notin \mathcal{M}$, is honest and reconstructs $F'$ for *ID*, then $F' = \perp$.

**Confidentiality:** Assuming there exists at least one honest client, the adversary plays the following game called CONF:

1. The adversary interacts with the honest parties in an arbitrary way.
2. The adversary outputs two files $F_0$ and $F_1$ such that $|F_0| = |F_1|$, the index $i$ of an honest client, an access list $\mathcal{L}$ not containing any corrupted servers, and a (unique) tag *ID*. Client $C_i$ (secretly) chooses a random bit $b$, and invokes the cDisperse protocol for storing the file $F_b$ with access list $\mathcal{L}$ and tag *ID*.
3. The adversary continues to interact with the honest parties subject only to the condition that no server in $\mathcal{L}$ starts a retrieval for *ID*.
4. The adversary finally outputs a bit $\hat{b}$.

The scheme satisfies *confidentiality*, if for all probabilistic polynomial-time adversaries playing the CONF game,

$$\mathsf{Pr}\left[b = \hat{b}\right] \leq \frac{1}{2} + \mathrm{negl}(\kappa),$$

where negl is some negligible function in the security parameter $\kappa$.

Recall that in our model the adversary is allowed to invoke protocols, and thus to trigger input actions for clients. Therefore, the condition in step 3 that no server in the access list $\mathcal{L}$ is allowed to retrieve the file that $C_i$ has stored with *ID* is necessary, since the adversary could otherwise start the retrieval for *ID* and trivially obtain the file.

### 2.3 Complexity measures

We assume that parties can erase memory in the complexity analysis. This is justified by the fact that the stored data has to be available over a longer time period and that we want to minimize the data stored by the servers after completion of a dispersal protocol.

The following complexity measures are used in the analysis of information dispersal schemes. Complexities are always defined with respect to a single instance of the scheme. (Recall that *associated messages* include only messages generated by honest parties.)

- The *message complexity* of a protocol is defined as the number of messages associated to an instance of the protocol.
- The *communication complexity* of a given protocol is defined as the bit length of all messages associated to an instance of the protocol.
- The *storage complexity* of an information dispersal scheme is the overall bit length of the information stored in the memory of the honest servers after they have completed the dispersal protocol.

It is clear that the information dispersal scheme must have storage complexity at least $|F|$ when a file $F$ is dispersed. The same holds for the communication complexity of the dispersal and the retrieval protocols. We therefore also use the following, more compact complexity measures:

- The *storage blow-up* of an information dispersal scheme is the ratio of the storage complexity of the dispersal protocol and $|F|$.
- The *communication blow-up* of a protocol (such as the dispersal or the retrieval protocol) that is part of an information dispersal scheme is the ratio of the communication complexity of the protocol and $|F|$.

The same definitions apply to schemes with confidentiality. Since we must always be able to reconstruct the original data from the information provided by $n - t$ servers, the storage blow-up cannot be smaller than $\frac{n}{n-t}$.

## 2.4 Tools

In the following, we will often refer to the problem of *asynchronous reliable broadcast* in our system model (which is also known as the *Byzantine generals problem* [LSP82]). A protocol for reliable broadcast allows a sender to broadcast a message $m$ to all servers such that either all or no honest server delivers $m$, even if the sender is faulty. A formal definition is provided in Appendix A. The standard implementation for it is Bracha's protocol [Bra84], which requires $\mathcal{O}(n^2)$ messages and has communication complexity $\mathcal{O}(n^2|m|)$.

A *collision-resistant hash function* is a function $H : \{0,1\}^* \to \{0,1\}^h$ with the property that the adversary cannot generate two distinct strings $x$ and $x'$ with $H(x) = H(x')$, except with negligible probability. With a slight abuse of notation, we denote by $|H|$ the bit-size of the range of the hash function, that is, $|H| := h$. In practice, $H$ could be implemented by SHA-1 (in this case, $|H| = 160$).

A *symmetric cryptosystem* is a triple $\mathcal{SE} = (SG, SE, SD)$, where *SG* is a key-generation algorithm, and *SE* and *SD* are an encryption- and a decryption-algorithm, respectively, such that for all messages $m$ and all keys $K$ generated according to *SG*, $SD(K, SE(K, m)) = m$. Observe the following game played by the adversary, which we call $\mathsf{SS}$:

1. The adversary chooses two messages $m_0$, $m_1$, where $|m_0| = |m_1|$, and gives them to an *encryption oracle*.

2. The encryption oracle secretly chooses a secret key $K$ according to *SG* and a random bit $b \in_R \{0,1\}$. It then returns a *ciphertext* $c := SE(K, m_b)$ to the adversary.

3. The adversary computes a bit $\hat{b}$.

The symmetric cryptosystem $\mathcal{SE}$ is *semantically secure* if and only if for all probabilistic polynomial-time adversaries, $\mathsf{Pr}[b = \hat{b}] \leq \frac{1}{2} + \text{negl}(\kappa)$, where negl is some negligible function in the security parameter $\kappa$. Note that this simple security requirement will be sufficient for our purposes.

# 3 An information dispersal scheme

In this section, our main scheme for asynchronous verifiable information dispersal is presented. The first part of the section is devoted to a review of basic tools from coding theory that are needed in our scheme and we also show a first simple solution for asynchronous verifiable information dispersal. Then, the actual scheme, called $\mathsf{AVID}$, is introduced and analyzed. Several improvements and optimizations to the scheme are presented subsequently, which reduce its communication and storage complexities. It is also shown how to derive a communication-efficient protocol for asynchronous reliable broadcast directly from our scheme. Finally, a variant of our scheme based on error-correcting codes is sketched; its storage complexity depends on $n$ and $t$ only, but it has a smaller resilience.

## 3.1 Tools from coding theory

We briefly review some basic tools from coding theory we will need in our scheme for asynchronous verifiable information dispersal. We make use of $(k,n)$-*erasure codes* over a finite field $GF(q)$. Such a code maps a vector $\mathbf{m} \in GF(q)^k$ to a vector $\mathbf{c} \in GF(q)^n$ such that knowing $k$ components of $\mathbf{c}$ is enough in order to reconstruct $\mathbf{m}$. Despite some similarities, this is different from a $(k,n)$-*error-correcting code*, where the components of the vector $\mathbf{c}$ can also be altered. Erasure codes are error-correcting codes for erasure channels.

Rabin [Rab89] proposed a quite general approach for erasure coding. A very simple erasure-code is based on *polynomial interpolation*, and is indeed a special case of Rabin's IDA. Assume $q > n$. Given a subset $\mathcal{J} \subset \{1, \ldots, n\}$, $|\mathcal{J}| = k$, we define the corresponding *Lagrange coefficients* evaluated at

$x \in GF(q)$ as

$$\lambda_j^{\mathcal{J}}(x) := \prod_{\ell \in \mathcal{J} \setminus \{j\}} \frac{x - \ell}{j - \ell} \quad \text{for all } j \in \mathcal{J}. \tag{1}$$

Then, with $\tilde{\mathcal{J}} := \{1, \dots, k\}$, we can encode $\mathbf{m}$ as

$$c_i := \sum_{j \in \tilde{\mathcal{J}}} \lambda_j^{\tilde{\mathcal{J}}}(i) \cdot m_j \quad \text{for } i = 1, \dots, n. \tag{2}$$

Note that $c_i = m_i$ for $i = 1, \dots, k$ (such a code is called *systematic*). The code can be also seen as a $(k, n)$-linear code with generator matrix

$$\mathbf{G} := \left[ \begin{array}{ccc|ccc} 1 & & & \lambda_1^{\tilde{\mathcal{J}}}(k+1) & \cdots & \lambda_1^{\tilde{\mathcal{J}}}(n) \\ & \ddots & & \vdots & & \vdots \\ & & 1 & \lambda_k^{\tilde{\mathcal{J}}}(k+1) & \cdots & \lambda_k^{\tilde{\mathcal{J}}}(n) \end{array} \right] \in GF(q)^{k \times n}. \tag{3}$$

Thus, encoding can alternatively be done through a matrix multiplication $\mathbf{c}^T := \mathbf{m}^T \cdot \mathbf{G}$, while decoding from any $k$ components of $\mathbf{c}$ can then be done efficiently through Lagrange interpolation, because the components of $\mathbf{c}$ are points on a polynomial of degree at most $k - 1$, and since the components used are correct. When we say that *we interpolate a polynomial of degree* $k - 1$ *from a set with at least* $k$ *elements*, we mean to pick any $k$ elements in the set and to reconstruct the polynomial.

This code is actually a variant of a *Reed-Solomon Code*. It can be used for error correction as well. It achieves a minimal distance of $n - k + 1$, and according to the *Singleton bound*, this distance is optimal. Recall that an error-correcting code with minimal distance $2e + 1$ can correct up to $e$ errors. For the above code, a polynomial-time error-correcting procedure can be given, but special variants of Reed-Solomon codes allow for much more efficient error correction [Bla83]. In particular, for such a Reed-Solomon Code with minimal distance $d$, there exist efficient algorithms correcting $e$ errors *and* $s$ erasures, as long as $d \geq 2e + s + 1$.

A file $F$ can be modeled as a vector of length $k$ over some finite field $GF(q)$, where $q$ is either a prime or a prime power, so that it can be encoded by making use of a $(k, n)$-erasure code. Following the idea of Rabin [Rab89], each one of the $n$ servers receives one of the $n$ blocks of the encoded file. A drawback of this approach is that it requires $k \leq n - t$, since up to $t$ servers might refuse to provide their own blocks during retrieval. Thus, to store large files one has to choose a large $q$, which might be very inefficient and unpractical. A solution for this is the following technique, which we call *striping*. Assume a larger file is given, that is a vector $F \in GF(q)^{k'}$ for $k' > k$. We can split the file in $r := \lceil \frac{k'}{k} \rceil$ files $F^j$, $j = 1, \dots, r$, of length $k$ each. Then, each file $F^j$ is encoded to $G^j$ using a $(k, n)$-erasure code. Finally, we define each block of the encoding $G = [G_1, \dots, G_n]$ of $F$ as

$$G_i := \left[ G_i^1, \dots, G_i^r \right] \;\; i = 1, \dots, n. \tag{4}$$

Clearly, $k$ blocks of $G$ are enough for reconstructing $F$.

However, note that in the following we simply model files as vectors in $GF(q)^k$ for compactness reasons, even though it is not difficult to see that all results hold when striping is used.

### 3.2 A simple scheme

As mentioned in the section on related work, the protocol of Garay et al. [GGJR00] can easily be adapted to implement a simple asynchronous verifiable information dispersal scheme, which we call *asynchronous GGJR*. The idea is to replace the gateway by an *asynchronous reliable broadcast protocol* started by the client to send $F$, which makes the scheme robust against corrupted clients. When $F$ is delivered to a server, it applies an $(n - t, n)$-erasure code and computes the list of the hashes of all blocks using a collision-resistant hash function $H$. It keeps its own block in memory together with the list of hashes and erases all other blocks. We can use the Retrieve protocol given below with $k = n - t$ for retrieval. It is easy to see that this implements an asynchronous verifiable information dispersal scheme.

In the analysis of this and other schemes, we assume that the size of the files to be stored is large compared to $n$ and $|H|$, in particular that $|F| \gg n^2|H|$. This assumption is reasonable and $|F| > n^2|H|$ holds already for files $F$ of size larger than 8kB, for example, when $n = 20$ and $|H| = 160$. Under this assumption, the storage blow-up of the asynchronous GGJR scheme is asymptotically optimal, that is, equal to $\frac{n}{n-t} + o(1) \leq \frac{3}{2} + o(1)$. However, the communication complexity of its dispersal protocol is $\mathcal{O}(n^2|F|)$, which means that its communication blow-up is $\mathcal{O}(n^2)$.

### 3.3 The main scheme

The scheme presented in this section, called AVID, consists of two protocols Disperse and Retrieve for dispersal and retrieval, respectively. The dispersal protocol integrates the steps of broadcasting the file and computing the erasure code with each other. It is similar in this respect to the asynchronous verifiable secret sharing protocol of Cachin et al. [CKLS02]. We assume a collision-resistant hash function $H$ is given. Furthermore, every server maintains two *associative arrays*, *Data* and *Verify*, whose indices are tags *ID* and which can be accessed by all protocol instances. All unused entries in the arrays are initialized once with the value $\perp$.

A client $C_i$ wanting to disperse a file with tag *ID* encodes it as a vector $[F_1, \ldots, F_n]$ by using the $(k, n)$-erasure code of Section 3.1. Additionally, it computes the *fingerprints* of the data, which are represented by a vector $\mathbf{D} := [D_1, \ldots, D_n]$, where $D_j := H(F_j)$ for $j = 1, \ldots, n$. Such hash function-based fingerprints have been introduced for information dispersal by Krawczyk [Kra93].

A reliable broadcast protocol similar to the one of Bracha [Bra84] is then used to let the servers agree on the vector $\mathbf{D}$ and distribute $F$. In particular, $C_i$ initially sends to each server $P_j$ the corresponding block $F_j$, as well as the vector $\mathbf{D}$. Then, two rounds of message exchanges among all servers follow, in which every message sent from a server $P_i$ to a server $P_j$ contains, besides $\mathbf{D}$, the block $F_i$. A server accepts only messages containing blocks which are correct according to $\mathbf{D}$. In this way, every server sees enough blocks in order to reconstruct its own block (should it be missing) and to guarantee the consistency of all blocks. If the servers agree on some $\mathbf{D}$, then server $P_j$ stores it in *Verify*[*ID*] and stores the block $F_j$ in *Data*[*ID*]. Finally, it generates an output message and erases all local variables. In order to retrieve a file, a client has to make sure it receives correct blocks from enough servers that agree on some $\mathbf{D}$.

A detailed description of the Disperse and Retrieve protocols is given in Figures 1 and 2, respectively. We are going to prove the following theorem in the next section:

**Theorem 3.** *AVID is a $(k, n)$-asynchronous verifiable information dispersal scheme for $t + 1 \leq k \leq n - 2t$. For $k = n - 2t$, the storage blow-up of the dispersal protocol is bounded by $3 + o(1)$, while the communication blow-up is $\mathcal{O}(n)$ for the dispersal protocol and $\mathcal{O}(1)$ for the retrieval protocol.*

### 3.4 Analysis of the main scheme

The proof has structural similarity with the analysis of the AVSS protocol of [CKLS02], since both exploit similar ideas. We note that the proof also holds for an *adaptive adversary*, that is, an adversary that chooses up to $t$ servers to corrupt adaptively during the execution of the protocol.

For the reliable broadcast of the vector $\mathbf{D}$, we use the following standard lemma [Bra84]:

**Lemma 4.** *Suppose an honest server $P_i$ sends a* `ready`*-message containing $\mathbf{D}^{(i)}$ and a distinct honest server $P_j$ sends a* `ready`*-message containing $\mathbf{D}^{(j)}$, then $\mathbf{D}^{(i)} = \mathbf{D}^{(j)}$.*

We now prove that the scheme AVID satisfies the properties of Definition 1.

**Protocol** Disperse **for tag** *ID*

**initialization**: // Server $P_i$

**for all** $\mathbf{D}$ **do**

$e_{\mathbf{D}} := 0, r_{\mathbf{D}} := 0, \mathcal{A}_{\mathbf{D}} := \emptyset$

**upon** receiving $(\mathit{ID}, \mathtt{in}, \mathtt{disperse}, F)$: // Client $C_i$

compute a polynomial $f(x)$ of degree at most $k-1$ such that $f(j) = F_j$ for all $j \in [1, k]$

$\mathbf{D} := [H(f(1)), \ldots, H(f(n))]$

**for all** $j \in [1, n]$ **do**

send $(\mathit{ID}, \mathtt{send}, \mathbf{D}, F_j)$ to $P_j$

**upon** receiving a message $(\mathit{ID}, \mathtt{send}, \mathbf{D}, F_i)$ for the first time: // Server $P_i$

**if** $H(F_i) = D_i$ **then**

**for all** $j \in [1, n]$ **do**

send $(\mathit{ID}, \mathtt{echo}, \mathbf{D}, F_i)$ to $P_j$

**upon** receiving a message $(\mathit{ID}, \mathtt{echo}, \mathbf{D}, F_m)$ from $P_m$ for the first time: // Server $P_i$

**if** $H(F_m) = D_m$ **then**

$\mathcal{A}_{\mathbf{D}} := \mathcal{A}_{\mathbf{D}} \cup \{(m, F_m)\}$

$e_{\mathbf{D}} := e_{\mathbf{D}} + 1$

**if** $e_{\mathbf{D}} = \max\left\{\left\lceil\frac{n+t+1}{2}\right\rceil, k\right\}$ and $r_{\mathbf{D}} < k$ **then**

interpolate $\bar{f}(x)$ of degree at most $k-1$ from $\mathcal{A}_{\mathbf{D}}$

$\bar{F}_j := \bar{f}(j)$ for all $j \in [1, n]$

**if** $H(\bar{F}_j) = D_j$ holds for all $j \in [1, n]$ **then**

**for all** $j \in [1, n]$ **do**

send $(\mathit{ID}, \mathtt{ready}, \mathbf{D}, \bar{F}_i)$ to $P_j$

**else**

output $(\mathit{ID}, \mathtt{out}, \mathtt{abort})$

**upon** receiving a message $(\mathit{ID}, \mathtt{ready}, \mathbf{D}, F_m)$ from $P_m$ for the first time: // Server $P_i$

**if** $H(F_m) = D_m$ **then**

$\mathcal{A}_{\mathbf{D}} := \mathcal{A}_{\mathbf{D}} \cup \{(m, F_m)\}$

$r_{\mathbf{D}} := r_{\mathbf{D}} + 1$

**if** $e_{\mathbf{D}} < \max\left\{\left\lceil\frac{n+t+1}{2}\right\rceil, k\right\}$ and $r_{\mathbf{D}} = k$ **then**

interpolate $\bar{f}(x)$ of degree at most $k-1$ from $\mathcal{A}_{\mathbf{D}}$

$\bar{F}_j := \bar{f}(j)$ for all $j \in [1, n]$

**if** $H(\bar{F}_j) = D_j$ holds for all $j \in [1, n]$ **then**

**for all** $j \in [1, n]$ **do**

send $(\mathit{ID}, \mathtt{ready}, \mathbf{D}, \bar{F}_i)$ to $P_j$

**else**

output $(\mathit{ID}, \mathtt{out}, \mathtt{abort})$

**else if** $r_{\mathbf{D}} = k + t$ **then**

*Verify*[*ID*] $:= \mathbf{D}$

*Data*[*ID*] $:= \bar{F}_i$

output $(\mathit{ID}, \mathtt{out}, \mathtt{stored})$.

**Figure 1:** AVID - Dispersal protocol

**Protocol** Retrieve **for tag** $ID$

**upon** receiving $(ID, \texttt{in}, \texttt{retrieve}, ID')$: // Client $C_i$

  **for all** $j \in [1, n]$ **do**

    send $(ID, \texttt{retrieve}, ID')$ to $P_j$

  **wait for** $k$ messages $(ID, \texttt{block}, F'_j, \mathbf{D})$ from distinct servers

    with the same $\mathbf{D}$ and $D_j = H(F'_j)$

  let $\mathcal{J}$ be the set of servers from which such a $\texttt{block}$-message has been received

  interpolate a polynomial $f'(x)$ of degree at most $k-1$ from $\{(j, F'_j) : j \in \mathcal{J}\}$

  $F' := [f'(1), \ldots, f'(k)]$

  output $(ID, \texttt{out}, \texttt{retrieved}, F')$

**upon** receiving a message $(ID, \texttt{retrieve}, ID')$ from client $C_m$: // Server $P_i$

  **if** $Data[ID'] \neq \perp$ **then**

    send $(ID, \texttt{block}, Data[ID'], Verify[ID'])$ to $C_m$

**Figure 2:** AVID - Retrieval protocol

**Termination.** The termination property is trivially satisfied: if the dealer is honest, the only way one malicious server can potentially make an honest server not accept is by producing a collision with respect to $H$, which can happen with negligible probability only.

**Agreement.** We define a $\texttt{ready}$-message $(ID, \texttt{ready}, \mathbf{D}, F_m)$ from $P_m$ to be valid if and only if $D_m = H(F_m)$. Assume an honest server completes the dispersal protocol with $Verify[ID] = \mathbf{D}$. Then it has received at least $k+t$ valid $\texttt{ready}$-messages with the same $\mathbf{D}$. At least $k$ of these have been sent by honest servers. Since these $k$ honest servers have sent a $\texttt{ready}$-message to all servers, all honest servers receive at least $k$ valid $\texttt{ready}$-messages with $\mathbf{D}$. But then every honest server sends a valid $\texttt{ready}$-message to every other server, except with negligible probability. This can be seen as follows: if this is not true, then there has to be some honest server that aborts. Let $P_i$ be the server which completed the dispersal, and let $P_j$ be an honest server that aborts. Since $P_i$ never aborted, there is a polynomial $f^{(i)}(x)$ of degree at most $k-1$ such that $H(f^{(i)}(m)) = D_m$ for all $m = 1, \ldots, n$. But since $P_j$ indeed aborts and because of Lemma 4, there must be one $(m, F_m) \in \mathcal{A}_{\mathbf{D}}^{(j)}$ (the set $\mathcal{A}_{\mathbf{D}}$ of $P_j$) such that $f^{(i)}(m) \neq F_m$ and $H(F_m) = H(f^{(i)}(m))$. But this means the adversary has found a collision for $H$, which can only occur with negligible probability. In the end, every honest server receives $n - t \geq k + t$ valid $\texttt{ready}$-messages, and by Lemma 4, they all contain $\mathbf{D}$.

**Availability.** Since $k$ honest servers have accepted, they also hold the same verification information $\mathbf{D}$, and thus the client is always able to reconstruct some value, because $k \geq t + 1$.

**Correctness.** Let $\mathcal{J} \subset \{1, \ldots, n\}$ be the set of $k$ honest servers which complete the dispersal of $F$. We define

$$G_i := \sum_{j \in \mathcal{J}} \lambda_j^{\mathcal{J}}(i) \cdot \bar{F}_j \qquad \text{for } i = 1, \ldots, n.$$

Moreover, $G := [G_1, \ldots, G_k]$.

Assume an honest client has shared a file $F$ and $F \neq G$. Then every $\texttt{echo}$-message from an honest $P_i$ to an honest $P_j$ contains $\mathbf{D}$ and $F_i$ as computed by the honest client. If the servers in $\mathcal{J}$ computed their $\bar{F}_j$ from these $\texttt{echo}$-messages, then $\bar{F}_j = F_j$. But since $G \neq F$, there must be an honest server $P_j$ with $\bar{F}_j \neq F_j$. Thus, it must have received a value $F'_m \neq F_m$ from a corrupted server $P_m$ (either in an $\texttt{echo}$- or in a $\texttt{ready}$-message), which it accepted. Since clearly $H(F_m) = H(F'_m)$, the adversary has found a collision for $H$. But this can happen with negligible probability only.

For the second point, assume an honest client reconstructs a value $F' \neq G$ using some set $\mathcal{J}' \neq \mathcal{J}$ of $k$ servers. Since $k \geq t+1$, the value of $\mathbf{D}$ the client chooses must be the unique one held by the correct servers by Lemma 4. On the other hand, if $F' \neq G$, there has to be some value $F'_m$ received by some server $P_m$, $m \in \mathcal{J}' - \mathcal{J}$, with $F'_m \neq G_m$, but $H(F'_m) = D_m$. But we also have $H(G_m) = D_m$, except with negligible probability, since in order for a server $P_j$ in $\mathcal{J}$ to accept, its $\bar{F}_j$ must be on a polynomial $\bar{f}^{(j)}(x)$ of degree at most $k-1$: thus, in order for two honest servers to accept shares which are on different polynomials the adversary must have found a collision for $H$. Hence, if the second point is not satisfied, the adversary must have found a collision for $H$, either in the dispersal protocol or in the retrieval protocol.

**Complexity analysis.** As above, we assume that the size of the files to be stored is large compared to $n$ and $|H|$, in particular $|F| \gg n^2|H|$.

The message complexity of protocol Disperse is $\mathcal{O}(n^2)$ since the number of messages that every honest server sends to every other server is bounded by a constant. All messages have size $\frac{|F|}{k} + n|H|$, and thus the communication complexity of the dispersal protocol is $\mathcal{O}(n^2\frac{|F|}{k} + n^3|H|)$. The storage complexity of the scheme is $\frac{n|F|}{k} + n^2|H|$. For the case $k = n-2t$, we obtain a communication blow-up bounded by $\mathcal{O}(n)$ and a storage blow-up bounded by $3 + o(1)$. (Note that the storage blow-up is suboptimal because of $k \leq n-2t$.)

The message complexity of protocol Retrieve is clearly $\mathcal{O}(n)$. Since messages have size $\frac{|F|}{k}+n|H|$, the communication complexity for $k = n-2t$ is $\mathcal{O}(|F| + n^2|H|)$. Recalling that $|F| \gg n^2|H|$, the communication blow-up is $\mathcal{O}(1)$.

### 3.5 Reducing the storage and communication complexities

This section presents a more efficient information dispersal scheme called AVID-H, which reduces the storage and communication complexities with respect to AVID.

We previously assumed that $|F| \gg n^2|H|$ in the complexity analyses and have ignored the size of the hashes. But it makes sense to reduce the number of hash values that are used by the protocols and need to be stored. We base our approach on so-called *Merkle trees*, an approach which has also been suggested in by Alon et al. [AKK+04]. (That paper improved an earlier work of the same authors based on expander graphs [AKK+00].)

Assume $n = 2^l$ for some $l \in \mathbb{N}$ and that some hash function $H$ is given. The (binary) *hash tree* of a vector $[F_1, \ldots, F_n] \in GF(q)^n$ is a complete binary tree with $n$ leaves. To every node $v$ of tree, a value $val(v)$ is assigned. Each leaf $i$, for $i \in [1,n]$, gets the value $val(i) := H(F_i)$, while every inner node $v$ with children $v_1$ and $v_2$ receives the value $val(v) := H(val(v_1), val(v_2))$. Let now $i \in [1,n]$, then there is a unique path from the root $v_r$ of the hash tree to $i$, which we denote by $v_r = v_0, v_1, \ldots, v_l = i$ and which has length $l = \log n$. For every $j \in [1,l]$, let $w_j$ be the unique child node of $v_{j-1}$ such that $w_j \neq v_j$ (that is, $w_j$ is the unique *sibling* of $v_j$). Then we define the *fingerprint for a block* $F_i$ as

$$FP(i) := [val(w_1), \ldots, val(w_l)]. \tag{5}$$

Thus, for every $i \in [1,n]$, $F_i \in GF(q)$, fingerprint $FP = [h_1, \ldots, h_l]$ and value of the root $h_r$, we define a predicate verify$(i, F_i, FP, h_r)$ which is computed by the following code:

$h := H(F_i)$
**for** $j := l$ downto $1$ **do**
  $h := H(h, h_j)$ or $h := H(h_j, h)$ depending on the unique path from $i$ to $r$
**return** $h = h_r$

**Lemma 5.** *Let $H$ be a collision-resistant hash function. Assume two values $F_i$, $F'_i$ with fingerprints $FP$, $FP'$ are given such that* verify$(i, F_i, FP, h_r) =$ verify$(i, F'_i, FP', h_r) = \texttt{true}$ *for some common $h_r$, then $F_i = F'_i$, except with negligible probability.*

*Proof.* At least one collision on the path from $i$ to $v_r$ must be found. This can happen with negligible probability only. □

It is now clear that we can reduce the storage complexity if every server $P_i$ stores just the fingerprint $FP(i)$ and the value of the root hash $h_r$. It is less clear that also the communication complexity can be reduced, but it follows by replacing $\mathbf{D}$ by the root hash for providing agreement.

The improved scheme AVID-H is now obtained by modifying AVID as follows:

- The client, after encoding, computes the hash tree of $[F_1, \ldots, F_n]$.
- The underlying reliable broadcast protocol now is used to agree on the root hash value. In every `send`, `echo` and `ready`-message, the root hash and the fingerprint of the block are sent instead of $\mathbf{D}$.
- The index of the counters $r$, $e$ and of the set $\mathcal{A}$ is not $\mathbf{D}$ but the root hash.
- Checking of the hashes, i.e., the test that $H(F_i) = D_i$, is replaced by the verify predicate above. When a server reconstructs a $\bar{F}_j$, then it also reconstructs the corresponding hashes on the path from $j$ to the root, and uses then the verify-predicate.
- Only the fingerprint and the root hash value are stored apart from the data block. For retrieval, both are added to the `block`-message, and the client selects among the responses according to the root hash value.

It is easy to see that the proof of Theorem 3 can be adapted to the improved protocol, based on Lemma 5.

**Complexity analysis.** The message complexity is still $\mathcal{O}(n^2)$. On the other hand, every message has now size $\frac{|F|}{k} + (\log n + 1)|H|$. For this reason, the communication complexity is improved to $\mathcal{O}(n^2\frac{|F|}{k} + n^2 \log n|H|)$. Furthermore, the storage complexity is reduced to $n\frac{|F|}{k} + \mathcal{O}(n \log n|H|)$.

Choosing $k = n - 2t = \Theta(n)$, the communication complexity is $\mathcal{O}(n|F| + n^2 \log n|H|)$. Observe that $|F| = \mathcal{O}(k \log q)$, since every component can be encoded using $\mathcal{O}(\log q)$ bits. But because $q > n$, $|F| = \Omega(n \log n)$. Hence, by substituting the $n \log n$ term, the communication complexity can be written as $\mathcal{O}(n|F|)$, under the assumption that $|H|$ is a small constant. In this case, the communication blow-up is $\mathcal{O}(n)$.

### 3.6 A communication-efficient protocol for reliable broadcast

Given an asynchronous verifiable information dispersal scheme, it is possible to derive from it a protocol for asynchronous reliable broadcast: to broadcast a value $m$, the dealer starts the dispersal protocol for a file $F := m$, and when a server accepts, it immediately starts the retrieval protocol in order to deliver a value $m'$. Using the AVID-H scheme of the previous section, we obtain the following result:

**Theorem 6.** *Provided that $t < \frac{n}{3}$, there exists an asynchronous reliable broadcast protocol with message complexity $\mathcal{O}(n^2)$ and communication complexity $\mathcal{O}\left(n|m| + n^2 \log n|H|\right)$.*

Note that with the AVID-H scheme, only a slight modification of the Disperse protocol already provides reliable broadcast because the servers do not need to run the Retrieve protocol for delivering $m'$. Instead of storing one block of a file with the corresponding fingerprint, each server simply outputs the whole reconstructed file $\bar{F} = [\bar{F}_1, \ldots, \bar{F}_k]$ as the delivered message $m'$.

In Appendix A, this protocol is compared to related protocols for asynchronous reliable broadcast [Bra84, CKPS01].

### 3.7 Further optimizations

We present now two further optimizations for the AVID scheme. Note that these two ideas are mutually exclusive.

**Achieving the optimal storage blow-up.** In Section 3.2, we have explained how a simple asynchronous verifiable information dispersal scheme with asymptotically optimal storage blow-up can be derived from an asynchronous reliable broadcast protocol. Therefore, by making use of the reliable broadcast protocol provided by the Disperse protocol (see Section 3.6), instead of Bracha's protocol in the construction of Section 3.2, we can realize an asynchronous verifiable information dispersal scheme with the message and communication complexities of the AVID-H scheme of Section 3.5, but whose storage blow-up is asymptotically optimal. We call this scheme AVID-RBC.

A possible drawback of this approach is of computational nature: erasure-coding has to be applied twice to the data. In the first step, an $(n-2t, n)$-erasure code is used for the broadcast, and in the second step, an $(n-t, n)$-erasure code is used in order to achieve the optimal storage blow-up.

**Getting rid of the hashes for $t$ small enough.** In Section 3.5 we have seen how the communication complexity can be reduced to roughly $\mathcal{O}(n|F|)$. However, for what regards the storage complexity, one might object that the stored information for the hashes might still be large enough to cause some problems. In this section, we show a technique for eliminating the hashes completely at the cost of lower resilience. The storage blow-up is in general larger than when hashes are used, but the dependency on $|H|$ is completely eliminated.

The approach has been suggested in [Kra93] and can be translated to our setting with some care. We will make use of error correcting codes, for example of a Reed-Solomon code allowing efficient decoding when errors and erasures are present. Using the error correcting properties of such a code, missing blocks are considered to be erasures and altered blocks are corrected.

When retrieving the blocks from a server, we can only expect to get at most $n-t$ of them. Since the adversary can arbitrarily slow down the messages from $t$ honest servers, up to $t$ modified blocks may be received. Thus, up to $t$ errors *and* $t$ erasures must be corrected. This requires a code with a minimal distance of $n-k+1 \geq 2t+t+1$, that is $k \leq n-3t$ (which requires $t < \frac{n}{4}$, since $k > t$).

The improved scheme, called AVID-ECC, is obtained by modifying the AVID scheme as follows:

1. We set $k = n - 3t$, and use the same dispersal protocol, but encoding the data with an appropriate Reed-Solomon code. At the end, only the blocks are stored, and not the hashes/fingerprints.
2. The retrieval protocol is modified to wait for $n-t$ blocks, without any checking. After that, error correction is applied.

The scheme achieves a storage blow-up of $\frac{n}{n-3t} < 4$, which is constant and does not depend on any extra $o(1)$-term.

# 4 Achieving confidentiality

Up to here we have used cryptography only for the sake of making the protocol robust against Byzantine parties. The second requirement on information dispersal defined in Section 2 is *confidentiality*. This property could, at a first glance, be satisfied very easily. Let $\mathcal{SE} = (SG, SE, SD)$ be a symmetric cryptosystem. A client who wants to store a file $F$ can simply store $SE(K, F)$ using a key $K$ that it keeps to itself. However, this works only when no other client should ever retrieve the file; the client also has to make sure that the key is not lost. In practice, the key should be stored by the system therefore, and more sophisticated access policies are needed. Of course, it is not possible to simply store the key with the dispersal protocol directly, since this would violate confidentiality. Garay et al. [GGJR00] proposed a solution for this on which our scheme builds.

## 4.1 Threshold cryptography

The protocol of Garay et al. [GGJR00] assumes that every client is associated with a public key for a threshold public-key encryption scheme and that the corresponding secret key is shared among the servers. Although this design reduces the trust assumptions on the client, the protocol needs a homomorphic encryption scheme that is "blindable" and therefore insecure against chosen-ciphertext attacks. This is a consequence of the system model which mandates the use of a gateway for controlling access to the servers.

In our model, we take a simpler approach and associate a threshold public key with the group of servers such that only one decryption key is shared by the system. However, we need a non-malleable threshold cryptosystem, i.e., one with security against adaptive chosen-ciphertext attacks.

An $(k, n)$-threshold cryptosystem $\mathcal{E}$ consists of the following elements [SG02]:

1. A *probabilistic key generation algorithm* that generates a triple $(PK, VK, SK)$, where $PK$ is the *public key*, $VK$ the *verification key* and $SK = [SK_1, \ldots, SK_n]$ is the list of *private keys*.

2. A *probabilistic encryption algorithm* $E$ that takes as input a public key $PK$, a message $m$, and a *label* $\ell$, and outputs a ciphertext $c = E(PK, m, \ell)$.

3. A *probabilistic decryption-share algorithm* $D$ that takes as input a private key $SK_i$, a ciphertext $c$, and a label $\ell$, and outputs a *decryption share* $\sigma = D(SK_i, c, \ell)$.

4. A *share verification algorithm* $\mathsf{verify}$ that takes as input the verification key $VK$, a ciphertext $c$, a label $\ell$, and a decryption share $\sigma$, and outputs $\mathsf{verify}(PK, c, \ell, \sigma) \in \{\texttt{true}, \texttt{false}\}$.

5. A *combining algorithm* $\mathsf{combine}$ that takes as input the verification key $VK$, a ciphertext $c$, a label $\ell$, and a set $\mathcal{S}$ of $k$ decryption shares, and outputs a message $m = \mathsf{combine}(VK, c, \ell, \mathcal{S})$, or the symbol $\perp$.

Note that the ciphertext depends on $\ell$. An encryption of the same message with two different labels leads to two distinct ciphertexts. We assume that the label $\ell$ can be efficiently extracted from the ciphertext.

A threshold cryptosystem is *correct* if for any message $m$ and label $\ell$, given a ciphertext $c := E(PK, m, \ell)$ and a set $\mathcal{S}$ of $k$ valid decryption shares computed by $\sigma = D(SK_i, c, \ell)$, we have $m = \mathsf{combine}(VK, c, \ell, \mathcal{S})$.

In order to define security, consider the following game called $\mathsf{TCCA2}$, where an adversary in our system model statically corrupts $t < k$ servers.

1. The key generation algorithm is run by a trusted party.

2. The adversary interacts with the uncorrupted servers in an arbitrary fashion, feeding them ciphertexts and obtaining decryption shares.

3. The adversary chooses two cleartexts $m_0$ and $m_1$, where $|m_0| = |m_1|$, as well as a label $\ell$, and gives them to an *encryption oracle*. The oracle chooses a bit $b$ at random, encrypts $m_b$, and returns the resulting ciphertext $c$.

4. The adversary continues to interact with the uncorrupted parties, feeding them ciphertexts $c' \neq c$ and receiving decryption shares.

5. The adversary outputs a bit $\hat{b}$.

The threshold cryptosystem is *secure against adaptive chosen-ciphertext attack* if for any polynomial-time bounded adversary playing $\mathsf{TCCA2}$, $\mathsf{Pr}[b = \hat{b}] \leq \frac{1}{2} + \mathrm{negl}(\kappa)$, where negl is a negligible function in the security parameter $\kappa$.

We can make use of the threshold cryptosystem proposed by Shoup and Gennaro [SG02], which is secure against chosen-ciphertext attack (in the random-oracle model), based on the hardness of the computational DH-Problem.

### 4.2 The scheme

We propose the following scheme, called cAVID, for asynchronous verifiable information dispersal with confidentiality. It consists of two protocols cDisperse and cRetrieve and is based on the scheme AVID from Section 3.

We use a symmetric cryptosystem $\mathcal{SE} = (SG, SE, SD)$ as well as a threshold cryptosystem $\mathcal{E}$, and assume that an honest dealer has set up the keys for the threshold cryptosystem, i.e., generated a triple $(PK, VK, SK)$ and shared $SK$ among the servers such that $P_i$ knows $SK_i$ for $i = 1, \dots, n$.

The cDisperse protocol works as follows. The client encrypts the file using the symmetric cryptosystem with an ephemeral key $K$ and encrypts $K$ using the threshold cryptosystem. It distributes the ciphertext of the file using the Disperse protocol and broadcasts the encrypted ephemeral key together with the access list $\mathcal{L}$ to the servers with a reliable broadcast protocol. During retrieval, every server sends a block of the file together with a decryption share for the ephemeral key to the client.

The details are given in Figures 3 and 4. The dispersal protocol calls the Disperse protocol of AVID and also uses a protocol RBC for reliable broadcast according to Section 2.4. However, using the AVID scheme has the advantage that the reliable broadcast protocol can be integrated with the Disperse protocol and does not cause any additional messages. We also stress that the proposed construction is general, and any other asynchronous verifiable information dispersal scheme can be used instead of AVID. Two additional associative arrays, *Key* and *Access*, are used and can be accessed by all protocol instances; they are also not erased at the end of the execution of the cDisperse protocol.

**Theorem 7.** *Provided $\mathcal{E}$ is a $(t+1, n)$-threshold cryptosystem that is secure against adaptive chosen-ciphertext attack, and provided $\mathcal{SE}$ is semantically secure, cAVID is a $(k, n)$-asynchronous verifiable information dispersal scheme with confidentiality for $t + 1 \le k \le n - 2t$.*

*Proof.* The proof for *termination* and *agreement* follows directly from the one of the AVID scheme and the properties of reliable broadcast. Recall, in the following, that we assume the channels are authentic. For *availability*, observe that honest servers agree on a common access list, and therefore all honest servers will either be sending some value or they will be rejecting the retrieval by sending an `unauthorized`-message. The correctness conditions of the cryptosystems imply that the client can retrieve the file. *Correctness* also follows easily from the correctness conditions of the AVID scheme and the cryptosystems, as well as from the properties of reliable broadcast. Moreover, the authenticity of the channels guarantees that non-authorized clients are not able to retrieve a file.

To show *confidentiality*, assume (toward a contradiction) that there is an adversary $A$ finishing the CONF game guessing $b$ with non-negligible probability over $\frac{1}{2}$. Making use of a hybrid argument, we show that we are able to construct a simulator which finishes the TCCA2 game for $\mathcal{E}$ with a non-negligible advantage or we construct a simulator which terminates the SS game for $\mathcal{SE}$ with a non-negligible advantage. It is clear that *confidentiality* then follows.

Provided that for $A$, $\Pr[b = \hat{b}] \ge \frac{1}{2} + \epsilon$, where $\epsilon = \frac{1}{\text{poly}(\kappa)}$, we can infer by a standard argument that

$$\left| \Pr\left[\hat{b} = 1 | b = 1\right] - \Pr\left[\hat{b} = 1 | b = 0\right] \right| \ge 2\epsilon. \tag{6}$$

Let $\mathcal{D}(ID, i, \tilde{K}, \tilde{F}, \mathcal{L})$ be the event where client $C_i$ disperses the encrypted file $\tilde{F}$ and broadcasts the key $\tilde{K}$ and the access list $\mathcal{L}$, making use of tag $ID$. Let $K_b$ be such that $\tilde{F}_b := SE(K_b, F_b)$ for $b \in \{0, 1\}$, and let $K$ be chosen according to $SG$, but independently of $K_1$ and $K_2$. Furthermore, denote $p(\tilde{K}, \tilde{F}) := \Pr\left[\hat{b} = 1 \middle| \mathcal{D}(ID, i, \tilde{K}, \tilde{F}, \mathcal{L})\right]$. Equation 6 yields:

$$\begin{aligned} 2\epsilon &\le \left| p(\tilde{K}_0, \tilde{F}_0) - p(\tilde{K}_1, \tilde{F}_1) \right| \\ &\le \left| p(\tilde{K}_0, \tilde{F}_0) - p(\tilde{K}, \tilde{F}_0) \right| + \left| p(\tilde{K}, \tilde{F}_0) - p(\tilde{K}, \tilde{F}_1) \right| + \left| p(\tilde{K}, \tilde{F}_1) - p(\tilde{K}_1, \tilde{F}_1) \right|. \end{aligned} \tag{7}$$

One of the three terms of this last sum must be therefore at least $\frac{2}{3}\epsilon$. Hence, we can distinguish three cases:

**Protocol** cDisperse **for tag** *ID*

**upon** receiving $(ID, \texttt{in}, \texttt{disperse}, F, \mathcal{L})$: // Client $C_i$

$K := SG(1^\kappa)$
$\tilde{F} := SE(K, F)$
$\tilde{K} := E(PK, K, ID)$
disperse $\tilde{F}$ using the Disperse protocol with tag $ID|disp$
broadcast $(\tilde{K}, \mathcal{L})$ to all servers using protocol RBC with tag $ID|bc$

**upon** delivering $(\tilde{K}', \mathcal{L}')$ from protocol RBC with tag $ID|bc$
and having completed the dispersal $ID|disp$: // Server $P_i$

**if** $Key[ID] = Access[ID] = \perp$ **then**
$Key[ID] := \tilde{K}'$
$Access[ID] := \mathcal{L}'$
output $(ID, \texttt{out}, \texttt{stored})$
**else**
output $(ID, \texttt{out}, \texttt{abort})$

**Figure 3:** cAVID - cDisperse protocol

**Protocol** cRetrieve **for tag** *ID*

**upon** receiving $(ID, \texttt{in}, \texttt{retrieve}, ID')$: // Client $C_i$

**for all** $j \in [1, n]$ **do**
send $(ID, \texttt{cretrieve}, ID')$ to $P_j$
**wait for** $k$ messages $(ID, \texttt{block}, \tilde{F}_j', \sigma_j, \mathbf{D}, \tilde{K})$ from distinct servers $P_j$
with the same $(\mathbf{D}, \tilde{K})$ such that $D_j = H(\tilde{F}_j')$, and $\mathsf{verify}(VK, \tilde{K}, ID', \sigma_j) = \texttt{true}$,
or $k$ messages $(ID, \texttt{unauthorized})$ from distinct servers
**if** $k$ block-messages have been received **then**
let $\mathcal{J}$ the set of $k$ servers for which a correct block-message has been received.
$\mathcal{S} := \{\sigma_j : j \in \mathcal{J}\}$
$K := \mathsf{combine}(VK, \tilde{K}, ID, \mathcal{S})$
interpolate a polynomial $\tilde{f}'(x)$ of degree at most $k-1$ from $\{(j, \tilde{F}_j') : j \in \mathcal{J}\}$
$\tilde{F}' := [\tilde{f}'(1), \ldots, \tilde{f}'(k)]$
output $(ID, \texttt{out}, \texttt{retrieved}, SD(K, \tilde{F}'))$
**else**
output $(ID, \texttt{out}, \texttt{retrieved}, \perp)$

**upon** receiving a message $(ID, \texttt{cretrieve}, ID')$ from client $C_m$: // Server $P_i$

**if** $m \in Access[ID']$ **then**
$\sigma_i := D(SK_i, Key[ID'], ID')$
send $(ID, \texttt{block}, Data[ID'|disp], \sigma_i, Verify[ID'|disp], Key[ID'])$ to $D_m$
**else**
send $(ID, \texttt{unauthorized})$ to $C_m$

**Figure 4:** cAVID - cRetrieve protocol

**Case 1:** $\left|p(\tilde{K}_0, \tilde{F}_0) - p(\tilde{K}, \tilde{F}_0)\right| \geq \frac{2}{3}\epsilon$.

We construct a simulator which achieves an advantage $\frac{\epsilon}{3}$ over $\frac{1}{2}$ for the TCCA2 game of the threshold cryptosystem $\mathcal{E}$. The simulator works as follows: during setup, it runs the key generation algorithm of $\mathcal{E}$. Then it simulates the CONF game to the adversary $A$ by controlling $n - t$ honest servers and any honest client. Whenever a decryption share is needed, the decryption is performed through a request to the corresponding honest server in the TCCA2 game. When $A$ chooses $F_0$ and $F_1$, $\mathcal{L}$, $ID$, and $i$, the simulator additionally chooses two keys $K_0, K_1$ independently and according to *SG*. It then sends $K_0$,$K_1$ and the label *ID* to the encryption oracle of the TCCA2 game, receiving a ciphertext $\tilde{K}_b$. Moreover, let $\tilde{F}_0 := SE(K_0, F_0)$. Now, the simulator simulates $C_i$ dispersing $\tilde{F}_0$ with tag *ID*$|disp$ and broadcasting $(\tilde{K}_b, \mathcal{L})$ with tag *ID*$|bc$. That is, the client $C_i$ chooses a random bit $b \in \{0, 1\}$. In case $b = 0$, $C_i$ stores the file $F_0$ regularly. On the other hand, if $b = 1$, $C_i$ stores the encryption on an independent key, which is different from the one being use to encrypt $F_0$ with $\mathcal{SE}$. Finally, when $A$ outputs $\hat{b}$, the simulator outputs $\hat{b}$. Observe that the guarantee of the TCCA2 game is satisfied, since a retrieval for *ID* is never started, and the tag *ID* is unique for every dispersal.

**Case 2:** $\left|p(\tilde{K}, \tilde{F}_0) - p(\tilde{K}, \tilde{F}_1)\right| \geq \frac{2}{3}\epsilon$

For this case, we construct a simulator which achieves an advantage of $\frac{\epsilon}{3}$ over $\frac{1}{2}$ in the SS game for $\mathcal{SE}$. The simulator works as follows: it simulates $n - t$ honest servers and all honest clients in the CONF game to $A$. It first initializes the threshold cryptosystem $\mathcal{E}$, acting as the honest dealer. When $A$ chooses two messages $F_0$, $F_1$, a tag *ID* and an access list $\mathcal{L}$, the simulator feeds them to the encryption oracle of the SS game, receiving a ciphertext $\tilde{F}_b$ back. The simulator also generates independently a random key $K$ and encrypts it using the public key of $\mathcal{E}$ with label *ID*, obtaining the ciphertext $\tilde{K}$. It simulates then client $C_i$ dispersing $\tilde{F}_b$ with tag *ID*$|disp$ and broadcasting $(\tilde{K}, \mathcal{L})$ with tag *ID*$|bc$. That is, for both values of $b$, the client $C_i$ encrypts a different, independently chosen key $K$ instead of the key used for the encryption of $F_b$. Finally, when $A$ outputs $\hat{b}$, the simulator outputs $\hat{b}$.

Observe that in this case only semantical security against passive attacks is needed by $\mathcal{SE}$, since at every use of the symmetric cryptosystem a new secret key is generated.

**Case 3:** This case is symmetrical to *Case 1*.

The desired advantages can in all cases be computed by a standard argument. □

**Complexity analysis.** The communication complexity depends on both the communication complexities of the disperse and the reliable broadcast protocols. We assume the reliable broadcast of $\tilde{K}$ and $\mathcal{L}$ takes place by appending the two values to every message of the Disperse protocol. In general, they are both very small, thus there is no need for a more efficient broadcast. For instance, the access list can be given through some succinct representation (say, through a small program), rather than as a large set of indices of clients. In particular, we assume $|F| \geq \max\{n^2|\tilde{K}|, n^2|\mathcal{L}|, n \log n|H|\}$. Assuming $k = n - 2t$ and that we are using the scheme AVID-H of Section 3.5, the communication complexity of cDisperse is $\mathcal{O}\left(n^2(\frac{|F|}{k} + \log n|H| + |\tilde{K}| + |\mathcal{L}|)\right)$, and the communication blow-up of the cDisperse protocol is $\mathcal{O}(n)$. The storage complexity is $\mathcal{O}\left(n(\frac{|F|}{k} + \log n|H| + |\tilde{K}| + |\mathcal{L}|)\right)$, which means that the storage blow-up is $\frac{n}{n-2t} + o(1)$.

Furthermore, making use of the scheme AVID-RBC of Section 3.7, the storage blow-up can even be reduced to $\frac{n}{n-t} + o(1)$.

**Applications.** The cAVID scheme can be used in order to directly derive a protocol for *asynchronous verifiable dual-threshold secret sharing* (see [CKLS02]) for $t+1 \leq k \leq n-2t$ which is storage-efficient (that is, which leads to very short shares), using the same approach as [Kra94].

We assume now there is just a set of servers $\{P_1, \ldots, P_n\}$ (that is, each party can be server *and* client), and up to $t < \frac{n}{3}$ of them can be corrupted by the *static* adversary. A dealer wanting to share a secret simply invokes the protocol cDisperse for storing the secret, choosing $\mathcal{L}$ in such a way that every party is allowed to read a secret. Now it is clear that no coalition of at most $t$ servers can gain any information (in a computational sense) about the shared secret, but every set of at least $k$ servers has enough information in order to reconstruct the secret.

This leads to shares of size $\frac{|s|}{k} + l$, where $l$ is the size of the key chosen in the cDisperse protocol. The communication complexity is $\mathcal{O}(n^2 \frac{|s|}{k})$.

## References


[AKK+00] Noga Alon, Haim Kaplan, Michael Krivelevich, Dahlia Malkhi, and Julien P. Stern. Scalable secure storage when half the system is faulty. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1853 of *Lecture Notes in Computer Science*, pages 576–587. Springer, 2000.

[AKK+04] Noga Alon, Haim Kaplan, Michael Krivelevich, Dahlia Malkhi, and Julien P. Stern. Addendum to scalable secure storage when half the system is faulty. *Information and Computation*, 2004. To appear.

[Bla83] Richard E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, Reading, 1983.

[Bra84] Gabriel Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 154–162, 1984.

[CKLS02] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proc. 9th ACM Conference on Computer and Communications Security (CCS)*, pages 88–97, 2002.

[CKPS01] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols (extended abstract). In Joe Kilian, editor, *Advances in Cryptology: CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. Springer, 2001. Full version available from *Cryptology ePrint Archive*, Report 2001/006, `http://eprint.iacr.org/`.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. 28th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 427–437, 1987.

[GGJR00] Juan A. Garay, Rosario Gennaro, Charanjit Jutla, and Tal Rabin. Secure distributed storage and retrieval. *Theoretical Computer Science*, 243(1–2):363–389, 2000.

[GWGR04] Garth R. Goodson, Jay J. Wylie, Gregory R. Ganger, and Michael K. Reiter. Efficient Byzantine-tolerant erasure-coded storage. In *Proc. International Conference on Dependable Systems and Networks (DSN-2004)*, pages 135–144, 2004.

[Kra93] Hugo Krawczyk. Distributed fingerprints and secure information dispersal. In *Proc. 12th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 207–218, 1993.

[Kra94] Hugo Krawczyk. Secret sharing made short. In Douglas R. Stinson, editor, *Advances in Cryptology: CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 136–146. Springer, 1994.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1992.

[Rab89] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.

[SG02] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2):75–96, 2002.

# A Reliable broadcast and verifiable information dispersal

Given the model introduced in Section 2.1, a protocol for *asynchronous reliable broadcast* is a protocol where a client (called a *dealer*) *broadcasts* a message $m$ and all the servers may *deliver* a value $m'$. Such a protocol satisfies the following properties:

**Validity:** If an honest dealer broadcasts a message $m$, some honest server eventually delivers $m$.

**Agreement:** If some honest server delivers a message $m'$, then all honest servers eventually deliver $m'$.

**Authenticity:** Every honest server delivers at most one message $m$. Moreover, if the dealer is honest, $m$ was previously broadcast by the dealer.

**Termination:** If the dealer is honest, then all honest servers eventually deliver a message.

Note that in contrast to the usual definition of reliable broadcast, where the dealer belongs to the set of servers, the dealer is a client in our context. This modification does not actually cause any problems, and existing protocols for reliable broadcast can be easily adapted in order to satisfy this new requirement.

The standard protocol for asynchronous reliable broadcast has been presented by Bracha [Bra84]. When broadcasting a message $m$, this protocol has message complexity $\mathcal{O}(n^2)$ and communication complexity $\mathcal{O}(n^2|m|)$. Note that the message complexity is actually optimal, and we cannot expect to achieve anything better.

Bracha's protocol has been improved by Cachin et al. [CKPS01] using a hash function $H$, in order to reduce the communication complexity in an *optimistic setting*. That is, if messages among honest parties arrive in time and if the servers controlled by the adversary are not actively interfering with the execution of the protocol, the communication complexity is bounded by $\mathcal{O}\left(n|m| + n^2|H|\right)$, where $|H|$ is the size of the hash function output. On the other hand, in the worst case, that is, if the corrupted servers cheat actively and the network is slow, the communication complexity can be as high as $\mathcal{O}\left(n^2(|m| + |H|)\right)$, and no improvement with respect to Bracha's protocol is achieved.

In Section 3.6 we showed that a communication-efficient reliable broadcast protocol can be derived from the Disperse protocol of the AVID-H scheme for asynchronous verifiable information dispersal. This protocol has communication complexity $\mathcal{O}(n|m| + n^2 \log n|H|)$. In contrast to the optimistic protocol of Cachin et al. [CKPS01], the communication complexity does not depend on $t$. Therefore, our protocol has a much smaller communication complexity than the optimistic protocol in the worst case and in any case an only slightly higher complexity than the optimistic protocol in the best case.

As a final remark, note that the since asynchronous verifiable information dispersal can be derived from asynchronous reliable broadcast as explained in Section 3.2, and because the converse also holds (as shown in Section 3.6), the two problems are equivalent.