Scalable Secure Storage when Half the System is Faulty

Noga Alon[1], Haim Kaplan[1], Michael Krivelevich[1], Dahlia Malkhi[2], and Julien Stern[3]

[1] School of Mathematical Sciences, Tel Aviv University, Israel

[2] School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel

[3] Laboratoire de Recherche en Informatique, CNRS - Universite de Paris Sud, France

Abstract. In this paper, we provide a method to safely store a document in perhaps the most challenging settings, a highly decentralized replicated storage system where up to half of the storage servers may incur arbitrary failures, including alterations to data stored in them.
Using an error correcting code (ECC), e.g., a Reed-Solomon code, one can take n pieces of a document, replace each piece with another piece of size larger by a factor of $\frac{n}{n-2t}$ such that it is possible to recover the original set even when up to t of the larger pieces are altered. For t close to $n/2$ the space overhead of this scheme is close to n, and an ECC such as the Reed-Solomon code degenerates to a trivial replication code.
We show a technique to reduce this large space overhead for high values of t. Our scheme blows up each piece by a factor slightly larger than two using an erasure code which makes it possible to recover the original set using $n/2 - O(n/d)$ of the pieces, where $d \approx 80$ is a fixed constant. Then we attach to each piece $O(d \log n / \log d)$ additional bits to make it possible to identify a large enough set of unmodified pieces, with negligible error probability, assuming that at least half the pieces are unmodified, and with low complexity. For values of t close to $n/2$ we achieve a large asymptotic space reduction over the best possible space blowup of any ECC in deterministic setting. Our approach makes use of a d-regular expander graph to compute the bits required for the identification of $n/2 - O(n/d)$ good pieces.

1 Introduction

In order to safeguard a document, the most simple solution is to replicate it, and to store the different copies in different places. This method, however, has two main drawbacks. First, the integrity of multiple replicas is harder to maintain, and second the required storage space grows linearly with the number of copies. In this paper, we provide a method to safely store a document that addresses both issues. First, our method guarantees integrity against arbitrary alterations, even malicious ones, in up to half of the storage servers. Second, the storage costs remain reasonable even in large systems, composed of hundreds or thousands of servers.

1.1 Our contribution.

Our approach makes use of an erasure code (that can recover the information provided some pieces are lost but the ones that remain are required to be correct) and adds verification information to the code pieces. For our computations, we assume usage of IDA [Rab89] whose space blow-up is optimal, though other erasure codes, e.g. [LMS+97,AL96], may be employed for efficiency with a slight sacrifice in space overhead. Let n be the number of pieces. We arrange the pieces in a graph, called the *storage graph*, such that each piece is a vertex, and an edge exists between vertices when they cross verify each other. Each vertex stores fingerprint information that transitively verifies every vertex up to distance k away in the graph, where k is a parameter chosen at setup time. A fingerprint is a digest of fixed length representing the content of a piece. Fingerprints have the property that it is highly unlikely (and infeasible) to find two different pieces with the same fingerprint. Typically, a cryptographically secure hash function, e.g., SHA1 [SHA1], is used as a digest function. The transitive verification information takes only a factor k more space than a regular fingerprint of the adjacent pieces. Herein lies a large gain, since each vertex verifies a neighborhood in the graph of radius k, which grows exponentially. The total storage cost is $O(kdn)$, where d is a bound on the storage graph degree. When $kd \ll n$, this cost is a significant improvement over previous methods. The complexity of our recovery and storage algorithms is $O(kdn)$ in addition to the time required for decoding and encoding the erasure code of choice. Our algorithm needs to compute, in the worst case, only kdn digests. The range of parameters which will be of particular interest for us is when d is constant and k is $O(\log n)$.

The storage graph we employ has the property that even when up to $t < n/2$ of its vertices are removed, a sufficiently large component of size $\Theta(n)$ remains connected with diameter $\leq k$. For this, we make use of known constructions of expander graphs [LPS86] and prove that the required properties hold in them. That is, we prove that if up to $t < n/2$ vertices are removed from an expander like that in [LPS86], then there remains a component of size $n/2 - O(n/d)$ with diameter $O(\log n / \log d)$. This result is of independent interest and may have other applications. Furthermore it can be extended to a setup when more than half the vertices are removed.

The retrieval algorithm selects a vertex at random and collects all the vertices that are verified by it, by a simple breadth-first-search. We show that this selection procedure needs to be repeated only an expected constant number of times until it collects a linear set of correct vertices. The total number of fingerprinting computations is bounded by $O(dn \log n / \log d)$. The computation of fingerprints dominates the time overhead of our retrieval algorithm over the decoding complexity of the erasure code we use.

1.2 Related work and alternative approaches.

The most prevalent approach for achieving resilience to arbitrary server corruption is the state machine approach [Lam79,Sch90], which numerous systems employ. Using this approach in the context of secure file storage, every server stores a full replica of the file and processes every update on it. The alteration of data stored by servers can be masked by obtaining $t+1$ identical replicas, where t is presumed to be a bound on the total number of corrupted replicas. Unfortunately, this method has a high overhead in storing full copies of the file at each replica.

When alterations to stored data are not of concern, erasure codes solve the problem. For example, Rabin [Rab89] presents a solution, called Information Dispersal Algorithm (IDA), which allows to transform a document of size s into n pieces of size s/m, where m is a parameter chosen by the user, such that the document can be reconstructed from any m pieces. Since the total amount of space taken by m pieces is exactly s, the space overhead of IDA is clearly optimal. However, if any of the obtained pieces is altered, the integrity of the reconstructed document may be compromised. Moreover, a user obtaining such an erroneous document has no way of detecting that an error has occurred, and may simply return erroneous results undetectably.

To overcome this problem, it is necessary to add redundant information to pieces when they are stored, that indicates when some other pieces(s) are altered. A simple approach is to store a fingerprint of the entire document with each piece. To recover the file, first one gets the correct fingerprint from a majority of the pieces, and then checks combinations of pieces for a file with the same fingerprint. However, this may lead to prohibitive computations in searching for a right combination of unaltered pieces.

To obtain a feasible solution one could use an error correcting code (ECC). An ECC takes n pieces and blows up each piece with some additional information such that it is possible to recover all the pieces provided that $n-t$ are uncorrupted, for any $t < n/2$. The minimal space-blowup factor of any ECC is $n/(n-2t)$ where n is the number of pieces. There are well known ECCs that achieve this optimal space overhead, e.g., Reed-Solomon codes. Unfortunately, when t approaches $n/2$, the space blows up by a factor of n (pieces) and this degenerates to simple replication.

Instead of using an ECC on the pieces themselves one can apply it to a shorter sequence of digests of the pieces, thereby reducing the space overhead at the expense of getting only a probabilistic guarantee for recovery. For example, the Secure IDA method in [Kra93] computes a *fingerprint* for each piece, and stores the vector of fingerprints using an ECC. To recover the document, first the vector of fingerprints is recovered, and then each piece is checked against its fingerprint. The space blow-up factor for storing a document with this method is $n/(n-t)$ for the IDA pieces, and an additional space for pieces of the fingerprints vector, blown up by a factor of $n/(n-2t)$. Here, too, when t approaches $n/2$, the fingerprints vector is fully replicated. The space for the fingerprints vector depends only on n and the digest function used, and does not depend on the document length. Nevertheless, this space could be quite prohibitive when n is large. To illustrate this, suppose a file size is 1 Mega-Byte, fingerprints are 160 bits, $n = 1000$ and $t = 499$. Then Secure IDA stores roughly 1000×160 extra bits, or $\approx 20KBytes$, with every IDA piece of $1MB/(1000 - 499) \approx 2KBytes$.

We can reduce the large blowup factor of ECC (either on the pieces themselves or on the fingerprints) by using a list decoding algorithm. The general idea is to use an ECC which is able to correct less errors than the maximum possible. In case the number of errors is larger than what the ECC is capable of fixing the list decoding algorithm will generate a small list of possible decodings of which we will be able to choose the right one with high probability. Polynomial list decoding algorithms for Reed-Solomon codes have been recently discovered by Sudan [Su97]. More specifically, a Reed-Solomon code codes K blocks into N blocks, such that any two codewords differ in at least $N - K + 1$ blocks (the distance). If at most $(N - K)/2$ blocks are altered, there is a single codeword that is closest to the altered data (i.e., differs from it in fewer than $(N - K)/2$ different blocks). This is the highest error for which Reed-Solomon is guaranteed to retrieve the original document. As already mentioned, for this error rate to reach half the blocks, Reed-Solomon blows up a stored document by a factor close to its size, thus triviliazing to full replication.[1]

In case the number of errors is larger than half the code distance a Reed-Solomon code can be used to recover a *list* of all possible decodings. The number of possible decodings is constant as long as the number of errors is less than $N - \sqrt{NK}$ (see [GRS95]) and the problem of finding the list is known as the *list-decoding* problem. Using techniques introduced by Sudan [Su97] and subsequently improved in [RR00] and [GS99], such a list is produced with a randomized polynomial time algorithm. In its most efficient form [GS99], their scheme corrects up to $\lceil N - \sqrt{NK} - 1 \rceil$ errors. The scheme can be used to address our problem as follows: A document of length K blocks is encoded into $n = N = 4K$ blocks using a Reed-Solomon code. In addition, we store with each block a digest H of the full document. To retrieve the document when up to half of the blocks may be altered, we recover H from the majority, and employ Guruswami and Sudan's list-decoding method to obtain a list of possible decodings, which we compare against H to retrieve the original document with high probability. The space blow-up of this method is constant $(= 4)$. The drawback of this scheme is the complexity of the retrieval algorithm which employs rather complicated methods, such as polynomial factorization, and has complexity cubic in n.[2] By comparison, our retrieval method is simpler (using only hashing and comparisons), and runs in $O(n \log(n))$ time. We use a completely different approach whose building blocks may have other applications.

A comparison of the efficiency of our method when half the system may be faulty with the various known approaches is given in Table 1 below.

[1] In previous paragraphs we thought of the document as decomposed into n pieces where n is fixed and the encoding increases the piece size. Here we think of the piece size as fixed and the encoding translates K pieces to N.

[2] The method by Roth and Ruckenstein [RR00] has computation complexity $O(n^2 \log^2 n)$ but needs twice as much space.

Method	Space overhead per server	Store time†	Retrieve time†
Our method	$(2+\epsilon)s/n + O(\log n)$	$O(n \log n)$	$O(n \log n)$
Simple replication	s	none	none
Reed-Solomon	s	none	none
Secure IDA	$2s/n + O(n)$	$O(n)$	$O(n)$
List decoding	$4s/n$	none	$O(n^3)$

† In addition to underlying coding/decoding time of the corresponding erasure or error correcting code.

Table 1. Comparison of methods: Storing document of size s on n servers when up to $t = n/2$ may be faulty.

Going back to our basic motivation, the need for scalable and survivable storage is reinforced in numerous recent systems that support information sharing in highly decentralized settings. Examples are the Eternity service [And96], a survivable digital document repository, SFS [MK98], a secure file system for a wide area network, Fleet [MR99], a survivable and scalable data replication system, a Byzantine file system of Castro et al. [CL99], and IBM's Evault [GGJ97], a storage system that employs Rabin's IDA to achieve survivable storage with reasonable storage burden. The verification information stored in these systems to guard against possible alteration of pieces does not scale to large system sizes. Our methods are most suitable for all the systems mentioned above and others, where scaling is a necessity.

The methods presented in this paper are concerned with the integrity of file storage and retrieval. Other aspects of data security are orthogonal to ours. Specifically, methods for preserving the secrecy of file contents in replicated systems have been proposed, e.g., in [HT88,AE90], such that the collusion of up to t faulty servers cannot reveal the contents of the information stored. These methods use secret sharing techniques that can be combined with our approach to achieve secrecy.

2 Preliminaries

The goal of this work is to provide two functions, *Share* and *Reconstruct*. Function *Share* takes a document x and produces n pieces denoted by $Share(x, 1), \ldots, Share(x, n)$. Function *Reconstruct* recovers the document with high probability despite arbitrary alterations in up to a threshold $t = \lfloor \frac{n-1}{2} \rfloor$ of the pieces.

Our algorithms make use of a cryptographically secure hash function H (such as SHA1 [SHA1]). For any value v, in an unlimited range, $H(v)$ has fixed size (in bits). We assume that it is computationally infeasible to find two different values v and v' such that $H(v) = H(v')$. Typically, setting $|H|$ to 160 bits suffices to guarantee this today, e.g., with SHA1, and hence we will assume this.

We also use Rabin's IDA [Rab89]. At a high level, IDA takes a data-value x and converts it into n pieces, $IDA(x, 1), ..., IDA(x, n)$, such that recovery of

x is possible from any combination of m pieces, and such that the total space taken by every m components is precisely $|x|$ (optimal).

3 *Share*

The *Share* transformation takes a document x and produces n pieces $Share(x, 1)$, ..., $Share(x, n)$ to be stored on n corresponding servers. The goal of the *Share* transformation is to allow retrieval of the original document x despite arbitrary corruption of up to t of the pieces. To cope with such alterations, we will transform x into n pieces and store verification information on each piece in such a way that discrimination between correct and incorrect pieces can be achieved at a low cost, while maintaining a low storage overhead. The challenge is to minimize the storage requirements to enable our scheme to scale up to very large systems. The secrecy of the document is not the main concern, and can be added using standard methods.

Our solution first transforms x using IDA into n pieces, $IDA(x, 1)$, ..., $IDA(x, n)$, such that x can be restored from any subset of $(\frac{n}{2} - \epsilon n)$ pieces (ϵ will be determined in Section 3.1). To safeguard against alteration of pieces, we add to each piece verification information as follows. Pieces are arranged in a *store graph* $ST = (V, E)$ on n vertices (which will also be specified in Section 3.1). We denote the set of vertices adjacent to a vertex i in ST by $N(i)$. Each vertex in ST represents one piece, and it stores k levels of verification information. For every vertex i, we define level-ℓ verification information V_i^ℓ recursively as follows: $V_i^0 = IDA(x, i)$ and for $l \geq 1$

$V_i^\ell = \langle H(V_{j_1}^{\ell-1}), \ldots, H(V_{j_{|N(i)|}}^{\ell-1}) \rangle$ where $j_1 < \ldots < j_{|N(i)|}$ are the neighbors of i.

In other words the level j verification information stored with piece i is the tuple of hashes of the level $j-1$ verification information stored at its neighbors. Each piece stores k levels of verification information that intuitively verify the pieces up to distance k away from i in the graph. (The parameter k will be determined in the next section). In addition, it stores the hash of the whole file.

The total space taken by each piece of a document x stored with our method is at most

$$|H|(dk + 1) + |x|/(\frac{n}{2} - \epsilon n) ,$$

where d is the maximum degree of any node in ST. When $dk = o(n)$, we get a significant improvement over ECCs. Note also that since the space overhead is proportional to the product of d and k, we can trade increased degree with decreased diameter, and vice versa. Hence, to be useful for storage and retrieval, we need the storage graph ST to have the following features:

- **Low degree**: The degree of vertex i in ST determines the storage overhead of the i'th piece.
- **Good expansion**: The expansion of ST determines the number of vertices that are at distance k from any particular vertex or group of vertices, and hence, the number of vertices that can be verified by them.

During retrieval, up to t vertices of ST may be corrupted, and hence, some set D of edges incident with $t \leq \lfloor \frac{n-1}{2} \rfloor$ vertices are removed. Notice that we do not know who are the t corrupt vertices and get to see a graph after deleting only the edges in D that are a subset of the edges incident with those t corrupt vertices. Nevertheless, using k-transitive verification, we know that every neighborhood of diameter k is either all correct or all corrupt. Hence, our graph construction needs to guarantee that after the removal of the set D of edges incident with t vertices from ST, there remains a set of $\frac{n}{2} - \epsilon n$ good vertices that are connected with a low diameter. We proceed to show such a construction.

3.1 Determining ST

We consider the problem of finding a storage graph ST such that when an arbitrary set D of edges incident with a set of $t = \lfloor \frac{n-1}{2} \rfloor$ malicious vertices is deleted there is still a large component with small diameter in the remaining part. We handle this case by picking a graph such that after the deletion of any set of t vertices we are guaranteed to have a set of almost $n - t$ vertices connected with a small diameter, say k, where we stipulate that $k = O(\log n / \log d)$. In the following we show that well known expander graphs [LPS86] satisfy our requirements. Namely, after deleting an arbitrary set of $t = \lfloor \frac{n-1}{2} \rfloor$ vertices, the remaining set of vertices contains a subgraph of size $\frac{n}{2} - O(\frac{n}{d})$ and of diameter $k = O(\log n / \log d)$, where $d \geq 80$ is a constant. The main result proved in the remainder of this section is therefore as follows:

Theorem 1. *In an LPS expander [LPS86] with $d > 80$, if one deletes half of the vertices then there is a vertex w such that $n/2 - O(n/d)$ of the remaining vertices are at distance $O(\log n / \log d)$ from w.*

We shall use the following result of Alon et al. [AFWZ95].

Theorem 2. *Let $G = (V, E)$ be a d regular graph such that the absolute values of the eigenvalues of its adjacency matrix but the largest are no greater than λ. For a set $B \subseteq V$, $|B| = \mu |V|$, let P be the set of walks of length k (edges) that are all contained in B. Then,*

$$|B| d^k \left(\mu - \frac{\lambda}{d} (1 - \mu) \right)^k \leq |P| \leq |B| d^k \left(\mu + \frac{\lambda}{d} (1 - \mu) \right)^k .$$

Proof (of Theorem 1). Fix a set $B \subseteq V$, $|B| = \frac{1}{2} n$. For a vertex $v \in B$ denote by P_v the set of walks of length k that start at v and never leave B. If follows from the lower bound in Theorem 2 that there is a vertex $w \in B$ for which

$$d^k \left(\frac{1 - \frac{\lambda}{d}}{2} \right)^k \leq |P_w|. \tag{1}$$

Denote by C the set of vertices occurring on walks in P_w. We claim that if

$$k = \frac{\log n}{\log \left(\frac{1 - \frac{\lambda}{d}}{2\left(c + \frac{\lambda}{d}(1 - c)\right)} \right)}$$

then $|C| \geq cn$. Otherwise, $|C| < cn$, and from the upper bound in theorem 2 we obtain that

$$|P_w| < cnd^k \left(c + \frac{\lambda}{d}(1-c)\right)^k \tag{2}$$

Combining the lower bound in (1) and the upper bound in (2) we obtain that

$$k < \frac{\log n}{\log\left(\frac{1-\frac{\lambda}{d}}{2\left(c+\frac{\lambda}{d}(1-c)\right)}\right)},$$

in contradiction with our choice of k.

In particular, for $c = 3\left(\frac{\lambda}{d}\right)^2$ we obtain that there is a vertex $w \in B$ such that there are at least $3\left(\frac{\lambda}{d}\right)^2 n$ vertices within distance

$$k = \frac{\log n}{\log\left(\frac{1-\frac{\lambda}{d}}{6\left(\frac{\lambda}{d}\right)^2+2\frac{\lambda}{d}-6\left(\frac{\lambda}{d}\right)^3}\right)} \tag{3}$$

from w in B. If we take LPS expander then $\lambda = 2\sqrt{d-1}$. It is easy to check that for $\lambda = 2\sqrt{d-1}$ and $d > 80$, one has $\frac{1-\frac{\lambda}{d}}{6\left(\frac{\lambda}{d}\right)^2+2\frac{\lambda}{d}} > 1$. Therefore, we obtain that if G is an LPS expander with $d > 80$ then there is a vertex $w \in B$ such that $3\left(\frac{\lambda}{d}\right)^2 n$ of the vertices of B are at distance at most $O(\log n/\log d)$ from w in B. (Notice that the constant hidden by the big-O approaches 2 as d goes to infinity.)

From Lemma 2.4 in Chapter 9 of [ASE92] it follows that if between two sets B and C such that $|B| = bn$ and $|C| = cn$ there is no edge then

$$|C|b^2d^2 \leq \lambda^2 b(1-b)n,$$

so $bc \leq (\frac{\lambda}{d})^2(1-b)$.

From this we get the following consequences:

1. There must be an edge between any set of size $3\left(\frac{\lambda}{d}\right)^2$ and any set of size $\left(\frac{1}{2}-\frac{\lambda}{d}\right)n$ if $\lambda/d < 1/4$.
2. There is an edge between every two sets of size $\frac{\lambda}{d}n$.
3. There is an edge between any set of size $(\frac{1}{2}-\frac{\lambda}{d})n$ and any set of size e/d if $e \geq \left(\frac{\lambda^2}{d}\right)\left(\frac{1/2+\lambda/d}{1/2-\lambda/d}\right)$.

For LPS expanders with $d > 80$, we have that $\lambda/d < 1/4$ and furthermore the condition in 3 holds for $e \geq 11$. Therefore we obtain that the set of vertices within distance $k+3$ from w where k is defined as in 3 is of size at least $(1/2 - 11/d)n$. (Notice that e is smaller and goes to 4 as d goes to infinity.) $\square$

4 *Reconstruct*

The goal of the *Reconstruct* transformation is to take n pieces retrieved from a storage system, up to t of which may be arbitrarily altered from the originally stored values for some document x, and to return the original document x. That is, given a set of pieces $\{r_1, ..., r_n\}$ containing at least $n-t$ original pieces $\{Share(x, i_1), ..., Share(x, i_{n-t})\}$, we want that $Reconstruct(r_1, ..., r_n) = x$ (with high probability). Note that, we need to keep *Reconstruct* feasible as n grows despite the uncertainty concerning up to t of the pieces. Hence, we cannot exhaustively scan all combinations of $n-t$ pieces to find a correct one.

Consider the set $\{r_1, ..., r_n\}$ of retrieved pieces. Every unaltered piece r_i contains the original values $\{V_i^j\}_{j=0,...,k}, H(x)$ which were stored in it. We say that r_i, r_j are *consistent* if they are connected in ST and all levels of verification information in them are consistent. This set induces a subgraph R of ST, with all the edges between inconsistent vertices removed. Our construction of *Share* guarantees the existence of a connected component in R of size $n/2 - O(n/d)$ whose diameter is at most k. Our recovery algorithm finds this set with an expected linear number of steps. Here we let $N^1(I) = N(I)$ be the set of vertices adjacent to vertices in I in the subgraph R of ST. Also we denote by $N^k(I)$ the set of vertices within distance no greater than k to a vertex in I. We prove the following lemma about R:

Lemma 1. *Let I be a set of fully unaltered vertices. Then every vertex in $N^y(I)$, where $y \leq k$, has its first $k-y$ levels of information unaltered.*

Proof. By induction on y. For the basis of the induction, we examine the immediate neighborhood of I. Since the first k-levels of verification information in each $r_i \in I$ are unaltered, for each immediate neighbor $r_j \in N(I)$ the hash values $H(V_0^j), \ldots, H(V_{k-1}^j)$ stored by some $r_i \in I$ are unaltered and hence, (by the cryptographic assumption) $V_0^j, \ldots, V_{k-1}^j$ are unaltered in j.

For the induction step, assume that the lemma holds for $y' < y$. Hence, every vertex in $N^{y-1}(I)$ has its $k-(y-1)$-levels of verification information unaltered. But since $N^y(I) = N(N^{y-1}(I))$ by an argument as for the base case stated above using $I' = N^{y-1}(I)$ and $k' = k-(y-1)$, we obtain that each vertex in $N(I')$ has $k'-1 = k-(y-1)-1$ levels of verification information unaltered, as desired. □

As an immediate corollary of Lemma 1 we obtain that if K is a connected set in R of diameter no greater than k then either all the IDA shares in K are correct, or all vertices of K are altered.

4.1 The algorithm

The algorithm *Reconstruct* goes as follows:

1. Let $S = \{r_1, ..., r_n\}$.

2. Let h be the value of $H(x)$ that occurs in $\lceil \frac{n+1}{2} \rceil$ pieces in S.
3. Pick a node $r_i \in S$ at random;
4. If $|N^k(R, r_i)| < n/2 - n/d$ set $S = S \setminus \{r_i\}$ and go back to step 3.
5. Get all the pieces from $N^k(R, r_i)$, reconstruct a document $\hat{x}$ using `IDA` and check that $H(\hat{x}) = h$. If so, return $\hat{x}$ else set $S = S \setminus \{\{r_i\} \cup N^k(R, r_i)\}$ and go back to step 3.

As shown, the storage graph contains, after removing t faulty nodes, a connected component of size $\Theta(n)$ and diameter $O(\log n)$. Hence, in an expected constant number of steps, the retrieval algorithm above will terminate and return the correct response.

5 A secure storage system

The application context of our work is a secure storage system. The system consists of a set S of n *servers* denoted $s_1, ..., s_n$, and a distinct set of *clients* accessing them. Correct servers remain alive and follow their specification. Faulty servers however may experience arbitrary (Byzantine) faults, i.e., in the extreme, they can act maliciously in a coordinated way and arbitrarily deviate from their prescribed protocols (ranging from not responding, to changing their behavior and modifying data stored on them). Throughout the run of our protocols, however, we assume a bound $t = \lfloor \frac{n-1}{2} \rfloor$ on the number of faulty servers. Clients are assumed to be correct, and each client may communicate with any server over a reliable, authenticated communication channel. That is, a client c receives a message m from a correct server s if and only if s sent m, and likewise s receives m' from c iff c sent m'. Furthermore, we assume a known upper bound τ on the duration of a round-trip exchange between a client and a correct server, i.e., a client receives a response to message m sent to a correct server s within at most τ delay. In our protocols, we need not make any assumption nor employ communication among the servers.

The storage system provides a pair of protocols, *store* and *retrieve*, whereby clients can store a document x at the servers and retrieve x from them despite arbitrary failures to up to t servers. More precisely, the store and retrieve protocols are as follows:

store: For a client to store x, it sends a message $\langle \mathsf{store}, x \rangle$ to each server in S, and waits for acknowledgment from $n - t$ servers.

retrieve: For a client to retrieve the contents of x, it contacts each server in S with a request $\langle \mathsf{retrieve} \rangle$. It waits for a period of τ to collect a set of responses $A = \{a_s\}_{s \in S}$, where each a_s is either a response of the form $\langle \mathsf{piece}, x_s \rangle$, if s responded in time, or $\perp$ if the timeout expired before s's response was received. The client returns $\mathit{Reconstruct}(A)$ as the retrieved content.

Each server s_i that receives a message $\langle \mathsf{store}, x \rangle$ stores locally the value $Share(x, i)$. And when a server s receives a $\langle \mathsf{retrieve} \rangle$ request it promptly responds with the currently stored piece.

A few points are worthy of noting in this description. First, due to our failure model, a client may receive more than $n - t$ responses to a query, albeit some undetectably corrupted. By assumption, though, the retrieved set A will contain $n - t$ original pieces, and hence, our *Share* and *Reconstruct* algorithms above guarantee that *Reconstruct*(A) yields the original stored content of x. Second, the store protocol assumes that the computation of each piece is done by each individual server. This is done for simplicity of the exposition. Another possibility would be for a client or some gateway between the clients and servers to first compute the pieces $Share(x, 1), ..., Share(x, n)$ and then send each piece to its corresponding server. The latter form saves computation time by performing it only once at the client (or the gateway), and comes at the expense of increasing the load on the client during a *store* operation. Both forms can be supported (in a similar manner to [GGJ97]), and are orthogonal to the discussion here. Third, during a *retrieve* operation the client may optimize access to the servers, e.g., by contacting an initial set of $n-t$ servers, which will suffice in the normal faultless state of the system, and dynamically increasing it only as needed. Such methods are extensively discussed in the relevant literature on distributed systems and replicated databases, and are not the main focus of the work at hand. Finally, for simplicity, we have assumed that *store* and *retrieve* operations do not overlap, though in practice, concurrency control mechanisms must be applied to enforce this.

6 Discussion

Our research leaves open a number of issues. First, our constants, in particular, the degree d, are rather large, and hence the results are beneficial for very large systems only. We are looking for graph constructions facilitating our methods for smaller system sizes. One such family of candidates are finite projective geometries [A86].

Second, our adversarial assumption is rather strong, namely, fully adaptive malicious adversary, and it might be possible to improve efficiency if we adopt a weaker adversarial model. In particular, one might accept in practice a non-adaptive adversarial model, that is, one that gives the adversary t randomly chosen servers to corrupt. Early on in this work, we envisioned making use of a random graph–$G(n, p)$–in which each edge (i, j) exists with probability p. It is known that for such random graphs, connectivity occurs at $p = (\log n + \omega(n))/n$ (with diameter $d = O(\log n/\log\log n)$) and that the diameter becomes 2 at $p = \Theta(\sqrt{(\log n)/n})$ (See e.g. [Bollobas85]). Due to the independent selection of edges, any subgraph G' of $G(n, p)$, induced by removal of t randomly selected vertices, is itself a random graph. Hence, it is also connected with a small diameter. This provides a viable solution for smaller system sizes than our current results, albeit for the weaker adversarial model and while incurring an increased probability of error (that is, the probability that the resulting subgraph after removal of faulty vertices is not connected with small diameter). Other candidates to acheive better results in the weaker adversarial model are random regular graphs.

Acknowledgement We are thankful to the following people for many helpful discussions: Michael Ben-Or, Nati Linial, Eli Shamir and Rebecca Wright.

References

[AE90] D. Agrawal and A. El Abbadi. Integrating security with fault-tolerant distributed databases. *Computer Journal* 33(1):71–78, February 1990.

[A86] N. Alon. Eigenvalues, geometric expanders, sorting in rounds, and Ramsey theory. *Combinatorica* 6(3):207–219, 1986.

[AFWZ95] N. Alon, U. Feige, A. Wigderson and D. Zuckerman. Derandomized graph products. *Computational Complexity* 5:60–75, 1995.

[AL96] N. Alon and M. Luby. A linear time erasure-resilient code with nearly optimal recovery. *IEEE Transactions on Information Theory* 42:1732-1736, 1996.

[ASE92] N. Alon, J. Spencer and P. Erdos. The Probabilistic Method. John Wiley & Sons, Inc. 1992.

[And96] R. J. Anderson. The Eternity Service. In *Proceedings of Pragocrypt '96*, 1996.

[Bollobas85] B. Bollobás. *Random Graphs*, Academic Press, London, 1985.

[CL99] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In the *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, New Orleans, USA, February 1999.

[GGJ97] J. Garay, R. Gennaro, C. Jutla and T. Rabin. Secure distributed storage and retrieval. In M. Mavronicolas and P. Tsigas, editors, *11th Internation Workshop on Distributed Algorithms, WDAG '97*, pages 275–289, Berlin, 1997. (LNCS 1109).

[GRS95] O. Goldreich, R. Rubinfeld, and M. Sudan. Learning polynomials with queries: The highly noisy case. In *Proc. 36th IEEE Symp. on Foundations of Comp. Science*, pages 294–303. IEEE, 1995.

[GS99] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, September 1999.

[HT88] M. P. Herlihy and J. D. Tygar. How to make replicated data secure. In *Advances in Cryptology—CRYPTO '87 Proceedings* (Lecture Notes in Computer Science 293), pages 379–391, Springer-Verlag, 1988.

[Kra93] H. Krawczyk. Distributed fingerprints and secure information dispersal. In *Proceedings of the 12th ACM Symposium on Principles of Distributed Computing*, pages 207–218, 1993.

[Lam79] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocessor programs. *IEEE Transactions on Computers*, C-28(9):690–691, 1979.

[LPS86] A. Lubotzky, R. Phillips and P. Sarnak. Explicit expanders and the Ramanujan conjectures. In *Proceedings of the 18th ACM Symposium on the Theory of Computing*, pages 240–246, New York, 1986.

[LMS+97] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman and V. Stemann. Practical loss-resilient codes. In *Proceedings of the 29th Symposium on Theory of Computing*, May 1997.

[MR99] D. Malkhi and M. K. Reiter. An architecture for survivable coordination in large scale distributed systems. *IEEE Transactions on Knowledge and Data Engineering.* To be published.

[MK98] D. Mazières and M. F. Kaashoek. Escaping the evils of centralized control with self-certifying pathnames. In the *Proceedings of the 8th ACM SIGOPS European workshop: Support for composing distributed applications*, Sintra, Portugal, September 1998.

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

[RR00] R. M. Roth and G. Ruckenstern. Efficient decoding of Reed-Solomon codes beyond half the minimum distance. *IEEE Transactions on Information Theory*, to appear.

[Sch90] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys* 22(4):299–319, December 1990.

[SHA1] FIPS 180–1. Secure Hash Standard. NIST. US Dept. of Commerce, 1995.

[Su97] M. Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.