Scalable Secure Storage when Half the System is Faulty[§]

Noga Alon[*] Haim Kaplan[*] Michael Krivelevich[*] Dahlia Malkhi[†] Julien Stern[‡]

May 18, 2002

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 $\frac{n}{n-2t+1}$ 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 blowup factor of this scheme is close to n, and the overhead of an ECC such as the Reed-Solomon code degenerates to that of 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.

Our approach makes use of an *erasure code* that can recover the information when some pieces are lost but the ones that remain are guaranteed to be correct. We add verification information to the code pieces in order to identify a large collection of good pieces when we reconstruct the file. Several erasure codes were suggested in the literature. We shall assume usage of the Reed-Solomon codes with Berlekamp-Welch decoding (see [GS92] for a description of this method), whose space blow-up is optimal, though

[4]Preliminary version of this paper appears as N. Alon, H. Kaplan, M. Krivelevich, D. Malkhi, and J. Stern, "Scalable Secure Storage when Half the System is Faulty", in Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP 2000), Geneva, Switzerland. pages 576–587.

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

[2]**Contact author:** School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel. dalia@cs.huji.ac.il

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

other erasure codes, e.g. [LMS+97, AL96], may be employed for efficiency with a slight sacrifice in space overhead. The verification information that we add to the pieces is organized in the form of fingerprints of particular arguments. We produce these fingerprints using a cryptographically secure hash function H, such as the conjectured collision-resistant SHA-1 [SHA1]. For any value v, in an unlimited range, $H(v)$ has fixed size (in bits). We denote this size by $|H|$. 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.

Let n be the number of pieces. We arrange the pieces in a graph, called the *store graph*, with n vertices, such that each piece corresponds to a vertex. An edge exists between vertices when they cross verify each other. A vertex v of degree d stores a collection of $d*k$ fingerprints that transitively verify every vertex up to distance k away from v in the graph, where k is a parameter chosen at setup time. The store graph is a carefully chosen expander graph where the neighborhood of radius k of any vertex grows exponentially. Herein lies a large gain. The transitive verification information takes only a factor k more space than a regular fingerprint of the adjacent pieces but verifies an exponentially larger collection of vertices. The total storage cost for the verification information is $O(kdn)$, where d is an upper bound on the maximum degree of a vertex in the store graph. 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 fingerprints. The range of parameters which will be of particular interest for us is when d is constant and k is $O(\log_d n)$.

The store 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], even maliciously, then there remains a component of size $n/2 - O(n/d)$ with diameter $O(\log n/\log d)$. This improves a somewhat weaker but similar result of Upfal [U94]. 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 $O(kdn)$ where $k = O(\log_d n)$. The computation of fingerprints dominates the time overhead of our retrieval algorithm over the decoding complexity of the erasure code we use.

The structure of the rest of this paper is as follows. Section 2 describes related previous work and several alternative approaches to the problem which we consider. Section 3 describes our storage and retrieval algorithms. Section 4 proves the properties we need to hold in the store graph. Section 5 gives some details about possible applications of our methods. We conclude in Section 6.

2 Related work and alternative approaches.

Replication is the most simple approach for achieving resilience to arbitrary corruption of storage. Using this approach, the alteration of data stored by replicated 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 copy of the file at each server.

When alterations to stored data are not of concern, erasure codes solve the problem. For example, Reed-Solomon codes with the Berlekamp-Welch decoding (described in [GS92], and used in Rabin's IDA scheme [Rab89]) can be used as follows. We split the file into $n-t$ segments, each consisting of $s/(n-t)$ bits, and think of these segments as representing a polynomial of degree $n-t-1$ over a finite field F of cardinality $p \geq 2^{s/(n-t)}$. The n pieces to store are the values of this polynomial at n fixed points of the field F. Clearly we can reconstruct this polynomial from only $n-t$ such values. Since the total amount of space taken by $n-t$ pieces is exactly s, the space overhead 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 that can handle alterations in at most t pieces one could use an error correcting code (ECC). The terminology here is usually quite different. An error correcting code C of block length N, rate K, and distance D over a q-ary alphabet Σ, denoted by $[N, K, D]_q$, is a mapping from Σ^K (the message space) to Σ^N (the codeword space) such that any pair of strings in the range of C differ in at least D locations out of N. Reed-Solomon codes are a classical and commonly used construction of error correcting code for which $D = N - K + 1$. The alphabet Σ for such code is a finite field F, where $|F| = q \geq N$. The message specifies a degree $K-1$ polynomial over F. The mapping C maps this polynomial to its evaluation at N distinct points of the field F. Since two distinct polynomials of degree $K-1$ agree in at most $K-1$ places it follows that $D = N - K + 1$. Efficient algorithms are known [Ber68] to recover the message from a codeword of a Reed-Solomon code despite errors in less than $D/2$ values.

Translating this to our setting, we would use an ECC with $N = n$ the number of desired pieces, $K = n - 2t + 1$, and $D = 2t$ over an alphabet which is a finite field F with $2^{s/(n-2t+1)}$ elements. Similarly to our use of erasure codes, we split the file into $n - 2t + 1$ pieces each of size $s/(n - 2t + 1)$. We think of these pieces as representing a polynomial of degree $n - 2t$ over F. The codeword consists of n evaluations of this polynomial. The space blowup of this method is $n/(n - 2t + 1)$, larger than with using erasure codes. However we are able to recover the file despite arbitrary alterations of t pieces. Note that when t gets close to $n/2$, the space blows up by a factor close to n. In particular, if $t = \lfloor \frac{n-1}{2} \rfloor$ this method has space blowup similar to fully replicating the file n times.

Instead of using an ECC on the pieces themselves one can apply it to a shorter sequence of fingerprints 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 pieces, and an additional space for pieces of the fingerprints vector, blown up by a factor of $n/(n - 2t + 1)$. Here, too, when t approaches $n/2$, the fingerprints vector storage is blown by a factor n. The space for the fingerprints vector depends only on n and the hash 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 piece of $1MB/(1000 - 499) \approx 2KBytes$.

An alternative approach that can reduce the large blowup factor of ECC (either on the pieces themselves or on the fingerprints) is to use recent polynomial time algorithms for the list decoding problem. Consider the Reed-Solomon codes we previously mentioned. As we described, when the number of errors is less than half the distance D of the code then there is a unique codeword closest to the message, and efficient algorithms for finding it. Therefore we picked a code such that t is no greater than half of its distance.

When there are more errors than half the distance of the code there may be more than one possible decoding. The problem of finding all possible decodings in such cases is called the *list decoding problem.* For Reed-Solomon codes the number of possible decodings is constant as long as the number of errors is less than $N - \sqrt{NK}$ (see [GRS95]). Using techniques introduced by Sudan [Su97] and subsequently improved in [RR00] and [GS99], one can produce this list with a randomized polynomial time algorithm. In its most efficient form [GS99], the algorithm can handle up to $\lceil N - \sqrt{NK} - 1 \rceil$ errors.

We could use this list decoding algorithm for our purposes as follows. We pick a Reed-Solomon code with $N = n$ and K such that $\lceil n - \sqrt{nK} - 1 \rceil \geq t$. So t now is greater than half the distance of the code. In particular if $t = \lfloor (n-1)/2 \rfloor$ then we can take $K = n/4$. The alphabet we use is of size $2^{s/K}$ where s is the size of the text we encode (that is it is either the size of the file itself or the size of the sequence of fingerprints of the pieces of the file). In addition, we store with each of the n pieces generated by the ECC, a hash H of the full document. To retrieve the document when up to t of the pieces are altered we first recover the correct hash H of the document from a majority. Then we employ Guruswami and Sudan's list-decoding method to obtain a list of possible decodings. Finally we pick the decoding whose hash is H.

The space blow-up of this method is constant ($= 4$). The drawback of this scheme is the complexity of the retrieval algorithm. This algorithm employs rather complicated methods, such as polynomial factorization, and has complexity cubic in n.[1] 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.

Method	Space 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 $t = (n-1)/2$ 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

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

techniques that can be combined with our approach to achieve secrecy.

3 Storing and retrieving a document

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 $share(x, 1)$, ..., $share(x, n)$ to be stored in n corresponding servers. 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.

3.1 *Share*

Our solution transforms x using Reed-Solomon ECC into n pieces, $ECC(x, 1)$, ..., $ECC(x, n)$, such that x can be restored from any subset of $(\frac{n}{2} - \epsilon n)$ pieces (ϵ will be determined in Section 4). To safeguard against alteration of pieces, we add to each piece verification information as follows. We arrange the n ECC pieces in a particular graph on n vertices, each piece corresponds to a vertex of the graph. We call this graph the *store graph* and specify it precisely in Section 4. We define k *levels of verification information* associated with each vertex in the store graph recursively. We denote the verification information of level l associated with vertex j by $V^l(j)$. The verification information of level 0 associated with vertex j is its ECC piece, i.e. $V^0(j) = ECC(x, j)$. For $1 \leq l \leq k$ we define

$$V^l(j) = \langle H(V_{j_1}^{l-1}), \ldots, H(V_{j_{|N(j)|}}^{l-1}) \rangle \ ,$$

where $N(j) = \{j_1, \ldots, j_{|N(j)|}\}$ is the set of neighbors of vertex j. In other words the level l verification information associated with vertex j is the tuple of hashes of the level $l-1$ verification information associated with its neighbors. Intuitively, $V^l(j)$ verifies vertices at distance at most l from vertex j. The information we store with server j, $share(x, j)$, consists of k levels of verification information associated with vertex j , $V^l(j)$, $0 \leq i \leq k$, (where the first level is the jth ECC piece). For every server j, $share(x, j)$, also contains the hash of the whole file, $H(x)$. In what follows we identify server j with vertex j of the store graph. Therefore we will refer to servers corresponding to vertices adjacent to vertex j in the store graph the neighbors of server j.

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

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

where d is the maximum degree of a node in the store graph. When $dk = o(n)$, we get a significant improvement over error correcting codes.

To get some intuition about our scheme suppose server u maliciously alters its ECC piece. The hash of the ECC piece of server u is part of the verification information of level one stored at the neighbors of u. Therefore by inspecting the verification information of level one stored at an honest server adjacent to server u we will be able to detect that server u altered its data. However, the servers adjacent to server u may collaborate with server u and maliciously alter their shares. In case all the neighbors of server u maliciously collaborate with server u we can still detect it using the verification information of level two of a server at distance two from u. Let y be such server and let z be the common neighbor of vertices y and u. To collaborate with server u vertex z had to change its verification information of level one to contain the hash of the modified ECC piece of vertex u. Therefore if y is honest the hash of the modified verification information of level one of z will be different from what is stored with the verification information of level two of y. In general if we get shares of a connected subgraph of vertices of diameter at most k, then we can detect that there exists an altered ECC piece of a server in the subgraph if the subgraph contain at least one honest server. To retrieve the document we will find a connected subgraph with sufficiently many good vertices. The store graph will guarantee that such a subgraph exists.

More generally, during retrieval, up to t servers may be corrupted, and hence, some set D of edges incident with $t \leq \lfloor \frac{n-1}{2} \rfloor$ vertices in the store graph 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.

The desired properties of the store graph are therefore as follows. To reduce the storage overhead we would like the maximum degree of the store graph to be small. We also want to guarantee safe retrieval using a small number of levels of verification information. In order to retrieve the document we will need to guarantee the existence of a large connected component of diameter at most k of honest servers despite arbitrary faults in t other servers. Since the space overhead is proportional to the product of d and k, we can trade a higher degree and expansion for smaller number of levels of verification information.

To achieve these properties, the store graph will be an expander graph where in a radius logarithmic from any server we have linearly many other servers. This way we get a large number of servers verifying any one particular server for a given number of levels of verification information. In Section 4 we proceed to show such a construction.

3.2 *Reconstruct*

The goal of the *reconstruct* transformation is to take n shares, $share(x,j)$, $1 \leq j \leq n$, retrieved from the servers, up to t of which may be arbitrarily altered (or missing altogether, in which case we consider them as 0) 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 $\{share(x,i_1), \ldots, share(x,i_{n-t})\}$, we want that $reconstruct(r_1, \ldots, r_n) = x$ (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. Each such piece, say r_j, presumably contains the k levels of verification information $V^l(j)$, $0 \leq l \leq k$, and $H(x)$. We say that r_i and r_j are *consistent* if the following conditions hold:

1. Vertices i and j are adjacent in the store graph.
2. For every $0 \leq l \leq k$, $V^l(j)$ contains the hash of $V^{l-1}(i)$ in the position corresponding to vertex i in its tuple.
3. For every $0 \leq l \leq k$, $V^l(i)$ contains the hash of $V^{l-1}(j)$ in the position corresponding to vertex j in its tuple.

We define the *retrieve graph* as a subgraph of the store graph in which we keep the edges only between consistent vertices. If i and j are adjacent in the store graph and honest then clearly they will be adjacent in the retrieve graph. Therefore the edges missing in the retrieve graph are adjacent with the t corrupt servers.

A critical property of the store graph which we prove in Section 4 is the following.

Property 3.1 *If we delete t vertices from the store graph then in the remaining graph there exists a connected subgraph with $\frac{n}{2} - \epsilon n$ vertices and diameter at most k.*

Property 3.1 guarantees the existence of a large connected component containing a large number of honest servers. Recall however that we obtain the retrieve graph from the store graph by deleting a subset of the edges incident to the corrupt servers and not the corrupt servers themselves. Therefore the large component of honest servers guaranteed by Property 3.1 may contain also corrupt vertices. We may also have a large component of corrupt servers. To identify the good servers we will also use the following lemma showing that all vertices at distance at most k in the retrieve graph from an honest server even if corrupt must have unaltered ECC pieces. It follows from this lemma that in a connected subgraph of diameter at most k of the retrieve graph either all ECC pieces (verification information of level zero) are correct or all servers are corrupt.

Let I be a set of vertices in the retrieve graph. We let $N^1(I) = N(I)$ be the set of vertices adjacent to vertices in I in the retrieve graph. 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 the retrieve graph.

Lemma 3.2 *Let I be a set of honest servers. 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 all 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 $H(V^0(r_j)), \ldots, H(V^{k-1}(r_j))$ stored with the verification information of levels $1, \ldots, k$ of some $r_i \in I$ are unaltered. Since r_i and r_j are connected in the retrieve graph, $H(V^i(r_j))$ matches the hash of $V^i(r_j)$ for $0 \leq i \leq k-1$. So by the cryptographic assumption on the fingerprints $V^0(r_j), \ldots, V^{k-1}(r_j)$ must be unaltered.

For the induction step, assume that the lemma holds for $y' < y$. Hence, every vertex in $N^{y-1}(I)$ has its first $k-(y-1)$-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. □

Property 3.1 says that a connected subgraph of honest servers of size close to $n/2$ and diameter k exists. We denote that subgraph by W. A random vertex belongs to W with probability close to $1/2$. Our algorithm picks a random vertex v and collects all vertices in $N^k(v)$. If indeed v belongs to W then all W is contained in $N^k(v)$ so $|N^k(v)| \geq \frac{n}{2} - \epsilon n$. Furthermore by Lemma 3.2 we know that the ECC share of every vertex in $N^k(v)$ is correct. Therefore using the ECC shares of the vertices in $N^k(v)$ we will recover the file. In case v is not in W it is possible that $|N^k(v)| < \frac{n}{2} - \epsilon n$. But another possibility is that $|N^k(v)| \geq \frac{n}{2} - \epsilon n$ and all the vertices in $N^k(v)$ maliciously changed their shares. Therefore we need a way to distinguish between a good and large neighborhood of an honest server and a bad and large neighborhood of a malicious server. To that end we use the hash, $H(x)$, of the whole file stored with each server. We recover $H(x)$ from each of $r_1, \ldots, r_n$ and if these values differ we decide on the correct value by taking the value that occurs most often. Since more than half of the servers are honest we are guaranteed to recover $H(x)$ correctly. Then we recover the file using a large neighborhood $N^k(v)$. If the hash of the recovered file matches $H(x)$ we know that v was honest. Otherwise we know that v is corrupt so we discard $N^k(v)$ and repeat this process. Here is a detailed description of our algorithm.

1. Let $S = \{r_1, \ldots, r_n\}$.
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 $|N^k(r_i)| < n/2 - \epsilon n$ set $S = S \setminus \{r_i\}$ and go back to step 3.

5. Get all the pieces from $N^k(r_i)$, reconstruct a document $\hat{x}$ using `ECC` and check that $H(\hat{x}) = h$. If so, return $\hat{x}$ else set $S = S \setminus \{\{r_i\} \cup N^k(r_i)\}$ and go back to step 3.

This algorithm has constant overhead over the complexity of recovering the file using ECC. Since the size of W is close to $\frac{n}{2}$ the expected number of iterations until we draw a vertex $v \in W$ and terminate is close to 2.

4 The store graph

We consider the problem of finding a store 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 given in Theorem 4.1 below. Upfal proved in [U94] a similar result with somewhat weaker numerical constants.

Theorem 4.1 *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 4.2 *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,*

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

We do not intend to reproduce the proof of the above statement here. However, an intuitive explanation of Theorem 4.2 can be helpful for the reader. Clearly, the total number of walks of length k starting in B is $|B|d^k$ (first choose a starting point in B, and then make k steps, each time choosing one of the d edges leaving a current vertex). If the edge distribution in G is uniform enough and the set B is large enough, one can expect a random walk to stay inside B with probability about $|B|/n = \mu$ at each step. Therefore, the probability that a random walk of length k stays inside B should be about μ^k. Theorem 4.2 supports this intuitive explanation quantitatively.

Proof.[of Theorem 4.1] Fix a set $B \subseteq V$, $|B| = \frac{1}{2}n$ (we assume to simplify the presentation that n is even). 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 4.2 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 4.2 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 [AS92] 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 $3\left(\frac{\lambda}{d}\right)^2 n$ 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 $\frac{e}{d}n$ 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 small and goes to 4 as d goes to infinity.) □

For the sake of the completeness of our presentation we provide now a description of the LPS graphs. Let p, q be primes congruent to 1 modulo 4 and such that p is a quadratic residue modulo q. As usual, denote by $PSL(2, Z_q)$ the factor group of the group of all two by two matrices over $GF(q)$ with determinant 1 modulo its normal subgroup consisting of the scalar matrices $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and $\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$. A well-known theorem of Jacobi asserts that there are precisely $p+1$ four-tuples $a = (a_0, a_1, a_2, a_3)$ such that a_0 is an odd positive integer, a_1, a_2, a_3 are even integers, and $a_0^2 + a_1^2 + a_2^2 + a_3^2 = p$. With each such vector a we associate the matrix $\gamma_a \in PGL(2, Z_q)$:

$$\gamma_a = \begin{pmatrix} a_o + ia_1 & a_2 + ia_3 \\ -a_2 + ia_3 & a_0 - ia_1 \end{pmatrix},$$

where i is a fixed integer satisfying $i^2 \equiv -1 (\mathrm{mod}\ q)$. By the assumption that p is a quadratic residue modulo q, all these matrices have their determinants a square in $GF(q)$ and therefore can be normalized so that they lie in $PSL(2, Z_q)$. The LPS graph $X^{p,q}$ is defined then as the Cayley graph of $PSL(2, Z_q)$ with respect to the above $p+1$ matrices. (The Cayley graph of a group G with respect to a symmetric set of elements S is the graph with vertex set G and edges (x, y) if $x = sy$ and $s \in S$). The graph $X^{p,q}$ is a $d = p+1$-regular graph on $n = q(q^2-1)/2$ vertices, whose eigenvalues, besides $p+1$, are at most $2\sqrt{p}$ in their absolute values. The proof of the last fact can be obtained by applying results of Eichler and Igusa concerning the so-called Ramanujan conjectures. We address the reader to [LPS86] for more details.

To summarize we can fix our store graph ST to be an LPS expander with $d > 80$. In this case the ϵ which we used in previous sections is $O(1/d)$ and the number of levels of verification we need is $k = O(\log_d n)$.

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 $u_1, ..., u_n$, 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 u if and only if u sent m, and likewise u 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 u within at most τ delay. In our protocols, we need not make any assumption on, 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 $A = \{a_u\}_{u \in S}$, where each a_u is either a response of the form $\langle \mathsf{piece}, x_u \rangle$, if u responded in time, or $\perp$ if the timeout expired before u's response was received. The client returns $reconstruct(A)$ as the retrieved content.

Each server u_i that receives a message $\langle \mathsf{store}, x \rangle$ stores locally the value $share(x, i)$. When a server u 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 is finite projective geometries [Bat97, 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. We have intuition that indeed, in this setting smaller system sizes may be workable. Our intuitive direction is as follows: We can make 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 $d = O(\log n / \log\log n)$), where $\omega(n)$ is an arbitrarily slowly growing function of 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. Other candidates to achieve better results in the weaker adversarial model are random regular graphs. These intuitive directions appear to provide a viable solution for smaller system sizes than our current results, though their exact details need to be further explored. In particular, it should be determined for such methods what is the probability that the resulting subgraph after removal of faulty vertices is not connected with small diameter.

Acknowledgement

We are thankful to the following people for many helpful discussions: Michael Ben-Or, Nati Linial, Eli Shamir and Rebecca Wright. We thank the anonymous reviewers for helpful comments on the presentation of this paper.

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.

[AS92] N. Alon and J. Spencer. The Probabilistic Method. John Wiley & Sons, Inc. 1992.

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

[Bat97] L. M. Batten. Combinatorics of finite geometries. Second edition. Cambridge University Press, Cambridge, 1997.

[Ber68] E.R. Berlekamp. *Algebraic Coding Theory*, McGraw Hill, New York, 1968.

[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 International Workshop on Distributed Algorithms, WDAG '97*, pages 275–289, Berlin, 1997. (LNCS 1109).

[GS92] P. Gemmell and M. Sudan. Highly resilient correctors for polynomials. *Information Processing Letters* 43(4):169–174, September 1992.

[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.

[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.

[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.

[U94] E. Upfal. Tolerating linear number of faults in networks of bounded degree. *Journal of Information and Computation*, 114:312–320, 1994.