Addendum to "Scalable Secure Storage when Half the System is Faulty"

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

Introduction. We consider the following problem. A file of size s bits is to be stored on n disks. Our failure model assumes that a potentially malicious adversary may choose after the file is stored less than half of the disks, and arbitrarily alter the data they store. The adversary is constrained in its computation power, which is assumed to be polynomial. Note that this disk-corruption model is different from the classical fault model for error correcting codes, in which individual bits experience faults independently. The goal is to store the file on the disks in such as way that recovery of the file is possible despite the corruption with high probability, where probability is over the choices of data alterations made by the adversary.

With guaranteed correct recovery, standard methods from coding theory indicate a lower bound of sn total storage bits (on all disks together). However, if we allow negligible probability of error, more compact schemes are possible (see a survey of known approaches in [1]).

More concretely, the probabilistic relaxation of recovery guarantee allows the use of a cryptographically secure hash function, such as the conjectured collision-resistant hash function SHA-1 [7], in order to probabilistically fingerprint data. This works as follows: During storage time, digests of certain data values are stored, such that a polynomially bounded adversary has negligible probability of consistently modifying the data without mismatching its digest. The probability of false-match is typically very small, e.g., 2^{-160} with SHA-1, and is not dependent on other system parameters like n. Therefore, from here on, we neglect this probability of error in our exposition, and simply say that if a certain data matches an (unaltered) pre-stored digest, then w.h.p. the data is unaltered.

The scheme we previously suggested in [1] [1] requires holding for the fingerprinting information a total storage of $O(n \log n)$ bits. Our scheme employs expander graphs for redundant cross-checking of fingerprint values. Some of the techniques in [1] may have other applications. In particular, Theorem [1] [4.1] demonstrates the robustness of an LPS expander [3] against deletion of half of the vertices, improving on the numerical constants of a similar result by Upfal in [8].

In this addendum, we demonstrate a simple solution, which is an application of Merkle hash-trees [4, 5]. The solution is asymptotically as efficient as the method in [1], works even for small n, and has much smaller (and no hidden) constants. In the remainder of this exposition, we describe the building blocks of the solution and combine them together to obtain the full scheme.

[1]School of Mathematical Sciences, Tel Aviv University, Israel.
[2]School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel. dalia@cs.huji.ac.il
[3]Cryptolog International SAS, France.

[1]For previous solutions, we refer the reader to the survey of related works in [1].

IDA. Denote the file to be stored by F, thus $|F| = s$ bits. Rabin has proposed in [6] an Information Dispersal Algorithm (IDA) for fault tolerant storage and for message dissemination. IDA makes use of Reed-Solomon codes with the Berlekamp-Welch decoding (described in [2]), to store F on n disks 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.

The IDA protects a file against loss of information on up to t disks. 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.

The main idea of combining fingerprinting with IDA is to somehow detect which pieces are corrupted using the fingerprints, discard them, and then apply IDA with the good ones. The remaining, uncorrupted pieces suffice to reconstruct the file.

Below, we denote by $IDA(F,i)$ the i'th piece produced by the IDA, to be stored on the i'th disk.

Digests. We produce fingerprints using a cryptographically secure hash function H, such as the conjectured collision-resistant SHA-1 [7]. 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 SHA-1, and hence we will assume this.

Merkle Trees. A Merkle hash-tree over the values $\{IDA(F,1), ..., IDA(F,n)\}$ is a binary tree, where the i^{th} leaf corresponds to the value $IDA(F,i)$ to be stored on disk i. For simplicity, assume $n = 2^\ell$. Each internal node is identified by a pair (i,j), where the level i is between $1..\ell+1$, and j ranges between $1..2^{\ell+1-i}$. The levels of the nodes are ascending from bottom to top, such that the level of a leaf is 1 and the level of the root is $\ell+1$. We associate a hash value with each node recursively as follows. The hash value associated with leaf i is the hash of $IDA(F,i)$. The hash associated with an internal node is the hash of the hashes of its children. Let $h(i,j)$ denote the hash value of node (i,j) then more precisely we have

1. $h(1,j) = H(IDA(F,j))$, for $j = 1..2^\ell$.
2. $h(i,j) = H(h(i-1,2j-1), h(i-1,2j))$, for $i = 2..\ell+1$, and $j = 1..2^{\ell+1-i}$.

We now define the fingerprint which is stored on disk i (in addition to the data piece $IDA(F,i)$). Let $\langle b_\ell, b_{\ell-1}, ..., b_1 \rangle$ be the binary representation of i using $\ell = \log_2 n$ bits (from most to least significant, going from left to right). There is a unique ascending path from the leaf i in the hash-tree to the root $(\ell+1, 1)$, namely, $(1,i), (2, \lfloor i/2 \rfloor), (3, \lfloor i/4 \rfloor), ..., (\ell+1, 1)$. In this path, for each k, the node $(k, \lfloor i/2^{(k-1)} \rfloor)$ is a child of $(k+1, \lfloor i/2^k \rfloor)$. The second child of $(k+1, i/2^k)$ is the node $(k, \lfloor i/2^k \rfloor) * 2 + (1 - b_{(k-1)})$. We store on disk i, together with $IDA(F,i)$, the hash values of all siblings of the nodes on the path from leaf i to the root, and $h(\ell+1,1)$ – the hash value of the root. More precisely, define $finger(i) = g(\ell+1,i), h(\ell+1,1)$, and store it on disk i, where $g(\cdot,\cdot)$ is constructed recursively as follows.

- $g(0,i) = IDA(F,i)$.
- $g(j,i) = g(j-1,i)\ ,\ h(j-1, \lfloor i/2^j \rfloor * 2 + (1 - b_{(j-1)}))$.

The verification predicate of an IDA share $IDA(F,i)$ against the root value $h(\ell+1,1)$ is naturally defined as the reverse of the above: we verify the share by calculating the hash values of the nodes along the path from leaf i to the root, using the appropriate hash value of siblings of nodes on this path, stored in $g(\ell+1,i)$, and compare it with $h(\ell+1,1)$.

The main property achieved by the use of a Merkle tree over the IDA shares is the following. For any index i, if a pair of data items (d_i, f_i) passes the above verification against $h(\ell+1,1)$, then w.h.p. we have $d_i \equiv IDA(F,i)$ and $f_i = finger(IDA(F,i))$, i.e., these are the original IDA share and its fingerprint.

Storage and Retrieval The storage of F is done by computing its IDA shares $IDA(F,1),\ldots,IDA(F,n)$, and the corresponding fingerprints $finger(1),\ldots,finger(n)$, and storing $IDA(F,i), finger(i)$ on disk i.

Retrieval is done by reading all disks, and recovering $h(\ell+1,1)$ from a majority. Then for each disk i, we perform the verification test against $h(\ell+1,1)$. If the calculated hash of the root equals to $h(\ell+1,1)$ then we use $IDA(F,i)$ to recover the file. After a majority of the pieces are verified for correctness, the file content is restored using IDA.

It is easy to see that w.h.p. all the IDA shares used in the retrieval are correct, and hence, w.h.p. the file is correctly retrieved by our scheme.

Complexity. The storage complexity of the scheme is $|H|(\log n + 1)$ bits of information per node not counting the IDA storage. Thus the total storage for fingerprinting verification is $O(n \log n)$ bits. The computation costs are all negligible compared with the erasure code manipulation.

References

[1] N. Alon, H. Kaplan, M. Krivelevich, D. Malkhi, and J. Stern. Scalable Secure Storage when Half the System is Faulty. *Information and Computation* 174:203–213, 2002

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

[3] 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.

[4] R. C. Merkle. Protocols for public key cryptosystems. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy*, pages 122–134, 1980.

[5] R. C. Merkle. A certified digital signature. *Advances in Crytology* (CRYPTO '89), LNCS 435, pages 218–238, 1990.

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

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

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