SafeCloud

Storage Architecture

D2.1

Project reference no. 653884

February 2016

European Commission

Horizon 2020
European Union funding
for Research & Innovation

Document information

Scheduled delivery	16.01.2017
Actual delivery	16.01.2017
Version	1.1
Responsible Partners	UniNE

Dissemination level

Public

Revision history

Date	Editor	Status	Version	Changes
22.12.2015	H. Mercier	Draft	0.1	Initial TOC at large
22.01.2016	P. Felber	Draft	0.2	Editing of D2.1
14.02.2016	P. Felber	Draft	0.3	Overall architecture in D2.1
17.02.2016	H. Mercier	Draft	0.4	Coding and entanglement in D2.1
17.02.2016	V. Schiavoni	Draft	0.5	Experimental testbed in D2.1
17.02.2016	P. Felber	Draft	0.6	Missing parts and finalisation of D2.1
23.02.2016	P. Felber	Draft	0.7	Addressed comments from reviewers
26.02.2016	H. Mercier	Draft	0.8	Addressed last reviewers' comments
28.02.2016	H. Mercier	Final	1.0	Final release
21.12.2016	P. Felber	Final	1.1	Revision after first review

Contributors

H. Mercier (UniNE): Introduction + overall editing
R. Barbi, D. Burihabwa, P. Felber, H. Mercier, V. Schiavoni (UniNE): technical content

Internal reviewers

R. Rebane (Cyber)
J. Paulo (INESC TEC)

Acknowledgements

This project is partially funded by the European Commission Horizon 2020 work programme under grant agreement no. 653884.

More information

Additional information and public deliverables of SafeCloud can be found at http://www.safecloud-project.eu.

Table of contents

Executive summary

This deliverable describes the SafeCloud storage architecture. The architecture provides three complementary and innovative solutions:

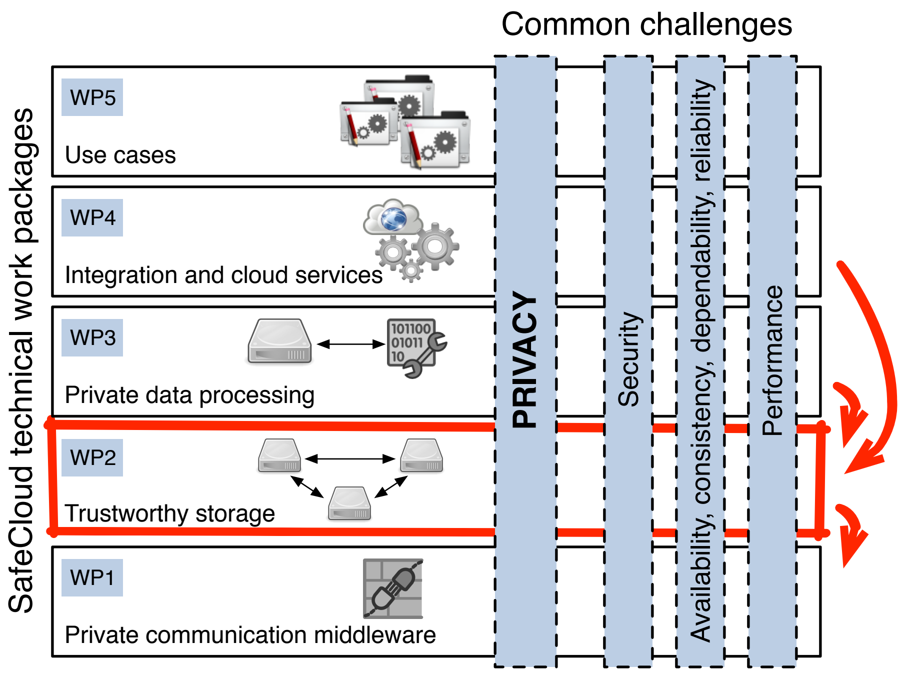
1. Secure block storage;
2. Long-term distributed encrypted document storage; and
3. Distributed encrypted file system.

These solutions differ in their objectives (data archival, data store), APIs (blocks, documents, POSIX file system), levels of protection and underlying mechanisms.

The deliverable introduces the general architecture of the solutions and provides details on the initial specification and implementation of the SafeCloud archival system. This long-term storage solution relies on innovative techniques for data coding and entanglement to provide strong anti-tampering capabilities and data integrity.

The deliverable also describes the SafeCloud storage testbed used for early prototyping and to gather initial insights into the performance and properties of coding and security libraries.

1 Introduction



Common challenges
SafeCloud technical work packages
WP5
Use cases
WP4
Integration and cloud services
WP3
Private data processing
WP2
Trustworthy storage
WP1
Private communication middleware
PRIVACY
Security
Availability, consistency, dependability, reliability
Performance

Figure 1: High-level architecture of the SafeCloud platform and dependencies between WP2 and other technical work packages.

This deliverable describes the storage architecture of the SafeCloud platform, built as part of work package 2. It leverages the secure communication mechanisms of work package 1 and, in turn, provides storage services to work packages 3 and 4 (see Figure 1).

The work package provides data storage mechanisms with an emphasis on privacy, data integrity and protection against tampering. In addition to these security features, the storage mechanisms must be designed with redundancy for protection against failures, and without significant performance degradation. Although the main storage contribution of SafeCloud is protection against tampering and censorship for long-term storage and permanent archival, the SafeCloud use cases require more storage granularity, including mutable data stores and file systems. We thus decided to design and provide three different data storage "solutions" for the SafeCloud platform.

This deliverable is organised as follows. In Section 2, we describe the motivation behind our work, and discuss the state of the art in Section 3. In Section 4, we describe how erasure-correcting codes can be used to provide high reliability and low storage overhead in distributed storage systems. The general storage architecture and its components, including our three storage solutions, are presented in Section 5. The most innovative of the three solutions, used for long-term storage and permanent archiving, is described in more details in Section 6. In Section 7, we describe the architecture of a first testbed prototype developed to drive the components, interfaces, and coding algorithms used for trustworthy data storage, and in Section 8 we discuss the implementation choices made for realizing this prototype. We conclude this deliverable in Section 9.

2 Motivation

As part of SafeCloud, we consider long-term digital data storage and permanent archiving. A first major challenge is data integrity. The objective is to provide verifiable guarantees to users that their data is properly, securely, and reliably archived. For instance, if a storage provider guarantees that the equivalent of three copies of each piece of data is archived on three continents, how can users verify that this claim is more than a marketing slogan? In practice, it appears difficult to prove this claim in a simple and convincing way. Users rely mostly in the good faith they have in their providers (and in the catastrophic consequences for their providers' bottom line should they lose the data). Another question is how can a user be sure that his data will not stop being taken care of after a system update or a maintenance budget cut, or that its data will be as securely and reliably stored as data from very large paying customers using the same service? These problems are especially relevant with old archives, some of which might not need to be accessed for decades.

A second challenge of digital storage and permanent archiving is tamper resistance. This is closely related to protection against censorship. Research and scientific data as well as medical, legal and financial records can include very sensitive information that can be viewed as threatening or compromising by potential censors. A good archival system must thus make it very difficult for a powerful censor to irrecoverably destroy or tamper with archived data, especially in an undetectable way. This is a different issue from the traditional definition of data integrity and authenticity, for which there already exists plenty of solutions from the client perspective using client-side cryptography.

For compliance and legal reasons, such sensitive data may be stored using a "write-once, read-many" (WORM) technology. WORM storage is a niche market of secure data storage solutions that has been historically fulfilled using hardware approaches. Physical implementations offer more constrained data access than logical approaches and are more dependent on hardware robustness against failures and destruction. The implementation of software approaches for anti-tampering is an active topic of investigation and no satisfying solution is available in practice.

There is currently no archival system providing strong anti-tampering and data integrity. Designing such a system is a surprisingly difficult endeavor, both in theory and in practice. This is one of the main objectives of SafeCloud.

3 State of the Art

Data integrity, protection against tampering and anti-censorship have been studied in various forms and for a large number of settings and applications. Work on these topics include the Eternity Service [And96], [WRC00], Freenet [CSW+01], Free Haven [DFM01], Dagster [SW01], Tangler [Wal01], SiRiUS [GSM+03], Tahoe [WW08] , Clouds [BHL+08] and POTSHARDS [SGM+09].

Randomized encryption can be used to prevent a malicious storage system from extracting information about its users by observing with whom they share files. It can also be used to prevent the system from preemptively censor documents corresponding to known content. This has been implemented with success in practice, notably by the Tahoe [WW08] filesystem under the name of Convergent Encryption. Encryption keys are semi-deterministically derived from the hash of a cleartext block so that efficient deduplication can still be performed on encrypted blocks to reduce storage space. However, a third party could publish the encryption key for a particular block, and prove that some block decrypts to censorable content, prompting an authority to individually censor particular blocks.

Data integrity and resistance against tampering are not easy to define, especially when considering how to provide these features in a practical way. For instance, in [WRC00], the authors informally write "Our system should make it extremely difficult for a third party to make changes to or force the deletion of published materials". This was also observed in [PRW05], which provided a more formal definition of censorship resistance in the context of selective filtering. A definition of data integrity was made in [AFY+07], but in such a strong way that it cannot be achieved in practical systems.

Other interesting related work include plausibly-deniable search [US12] and proofs of storage and retrievability [ZX12]. In [JK07] the authors describe an efficient proof of retrievability mechanism that allows a client to verify the existence of a piece of data in a storage system. The authors correctly note, however, that such a mechanism cannot guarantee that the system will agree to disclose the actual data when prompted to do so.

In [AFY+07], the authors studied data integrity and developed a theory of data entanglement. One of their contributions is the introduction of all-or-nothing integrity: intuitively, either all the documents are recoverable with high probability, or no document is. They show that all-or-nothing integrity is possible with some restrictions on the power of the attacker. [ADD+12] extended the work by providing a stronger definition of all-or-nothing integrity and a simulation-based security analysis. The protocols provided in both articles [AFY+07], [ADD+12] remain far from real-life implementations: they require to read the entire data store to retrieve a document, and require to process the entire data store to add a new document, which is not scalable. Furthermore, no document is recoverable if the storage provider corrupts or fails to maintain a small part of the data.

Providing anti-censorship using data entanglement was first proposed by Dagster [SW01] and Tangler [Wal01], and both can be seen as special cases of the approach we propose. In Dagster, documents and blocks have the same size. To add a new document in the system, c blocks already stored are chosen at random, and a new block consisting of the exclusive-or of the new document with the c blocks is stored. A censor wanting to delete a document can erase one of its $c+1$ blocks, and this will destroy on average $O(c)$ other documents, the older documents being more protected than newer ones. In Tangler, two old blocks chosen randomly and a new document to be archived are used to generate two new blocks using (3,4) Shamir secret sharing [Sha79]. The two new blocks are then stored. The original document can be recovered with any three of the four blocks using Lagrange interpolation. In [AFY+07], it was shown that erasing two blocks from a random Tangler document erases on average $O\left(\frac{logn}{n}\right)$ other documents. However, the number of documents erased irrecoverably is much smaller, since some partly corrupted documents can be decoded to recover erased blocks. No analysis of the system resistance against tampering is presented.

Finally, WORM storage has a long history predating CD-R disks and was commercialized in several forms for protection against tampering: tape cartridges, secure digital flash memory cards, SD cards, etc. Recent solutions include WORM HDDs, where the protection against data rewrite is embedded at the physical disk level [WOR].

4 Distributed storage using erasure correcting codes

The objective of error-correcting codes for data storage is to carefully add redundancy to data in order to protect it against corruption when stored on media like DVDs, magnetic tapes or solid-state drives. In these systems, the errors are usually modeled as erasures, meaning that their locations are known. Consider the example shown in Figure 2, where a coding disk is used to store the XOR of k data blocks.

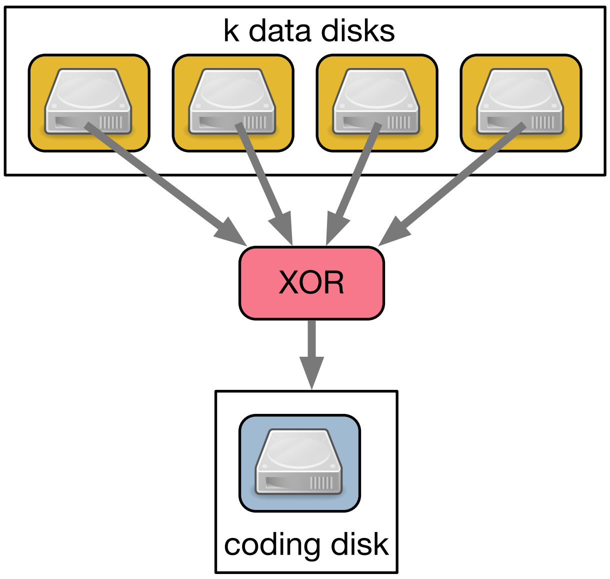


Figure 2: Redundancy using XOR.

If the system realizes that one of the disks has failed, as shown in Figure 3, it can XOR the healthy disks and recover the failure. This is the maximum decoding capability of this code, and there will be data loss if more than one disk fails.

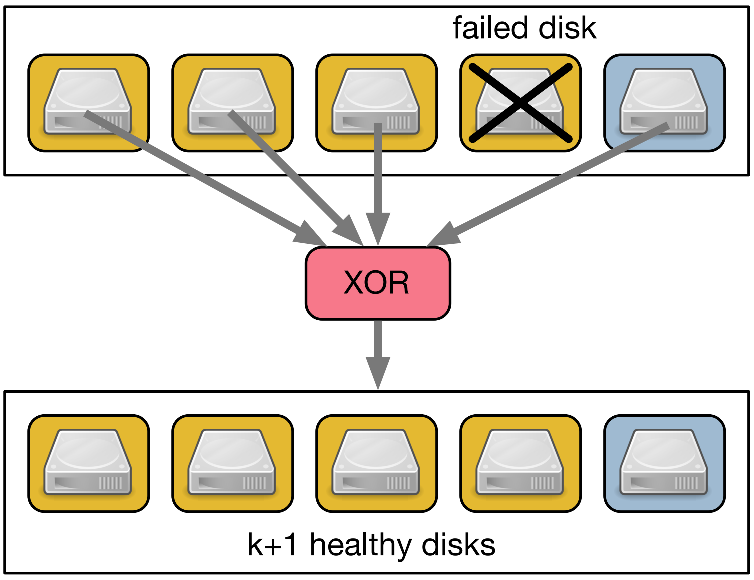


Figure 3: Failed disk recovered using XOR.

In general, k data blocks are coded to generate $n-k$ coding blocks, as illustrated in Figure 4.

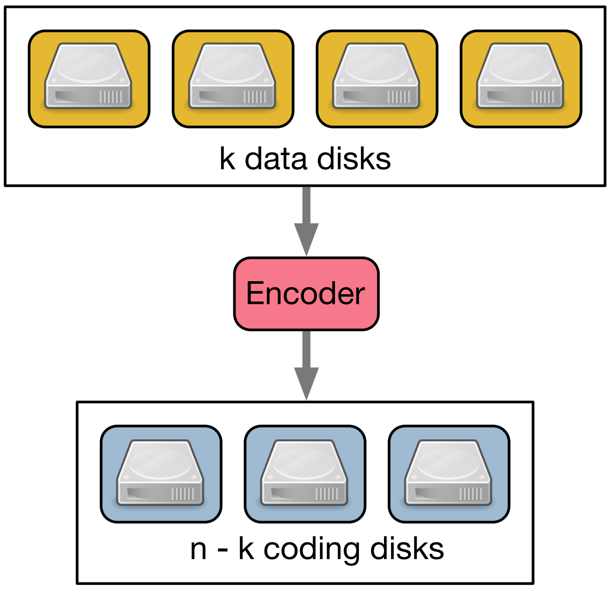


Figure 4: Generic erasure code encoder.

After disk failures, the system will try to decode the original codewords from the healthy disks, like in Figure 5. The number of recoverable disk failures depends on the code itself.

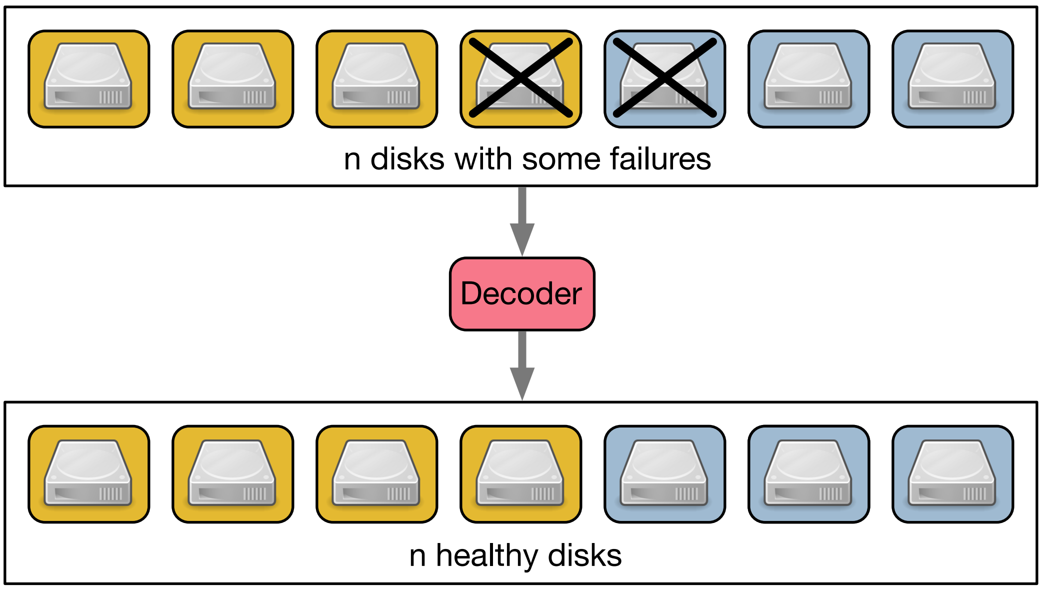


Figure 5: Generic erasure code decoder.

The most famous class of erasure codes are Reed-Solomon codes (RS), first introduced in 1960 [RS60]. An *(n,k)* Reed-Solomon code is a linear block code with dimension *k* and length *n* defined over the finite field of *n* elements. Reed-Solomon codes have many interesting properties. First, they achieve the singleton bound with equality, and thus are maximum distance separable (MDS) [LC04]. In other words, they can correct up to $n-k$ symbol erasures, i.e., any *k* of the *n* code symbols are necessary and sufficient for decoding. Second, using a large field, they can correct bursts of errors, thus their widespread adoption in storage media where such bursts are common.

Encoding and decoding Reed-Solomon codes is challenging, and optimizing both operations has kept many coding theorists and engineers busy for more than 50 years. There is a large amount of literature on these topics, covering theory (e.g., [GS99]) and implementation (e.g., [San00]), but in a nutshell the best encoding and decoding implementations are quadratic in the size of data.

Reed-Solomon, while storage-efficient, were not originally designed for distributed storage and are somewhat ill-suited for this purpose. Besides their complexity, their main drawback is that they require at least *k* geographically distributed healthy disks to recover a single failure, followed by decoding of all the codewords with a block on the failed disk. This incurs significant bandwidth and latency costs. This handicap has led to the development of codes that can recreate destroyed redundancy without decoding the original codewords. The tradeoffs between storage overhead and failure repairability is an active area of research [DGW+10], [OD15], and there are many interesting theoretical and practical questions to solve. Among other work of interest, NCCloud [CHL+14] reduces the cost of repair in multi-cloud storage if one cloud storage provider fails permanently. We also mention the coding work done for Microsoft Azure [CWO+11], [HSX+12], and XOR-based erasure codes [KBP+12] in the context of efficient cloud-based file-systems exploiting rotated Reed-Solomon codes. RAID-like erasure-coding techniques have been studied in the context of cloud-based storage solutions [KBP+12].

5 General architecture and components

The storage architecture provides three "solutions", in increasing levels of sophistication:

1. Secure block storage.
2. Long-term distributed encrypted document storage.
3. Distributed encrypted file system.

These three solutions are illustrated in Figure 6 and described in the rest of this section.

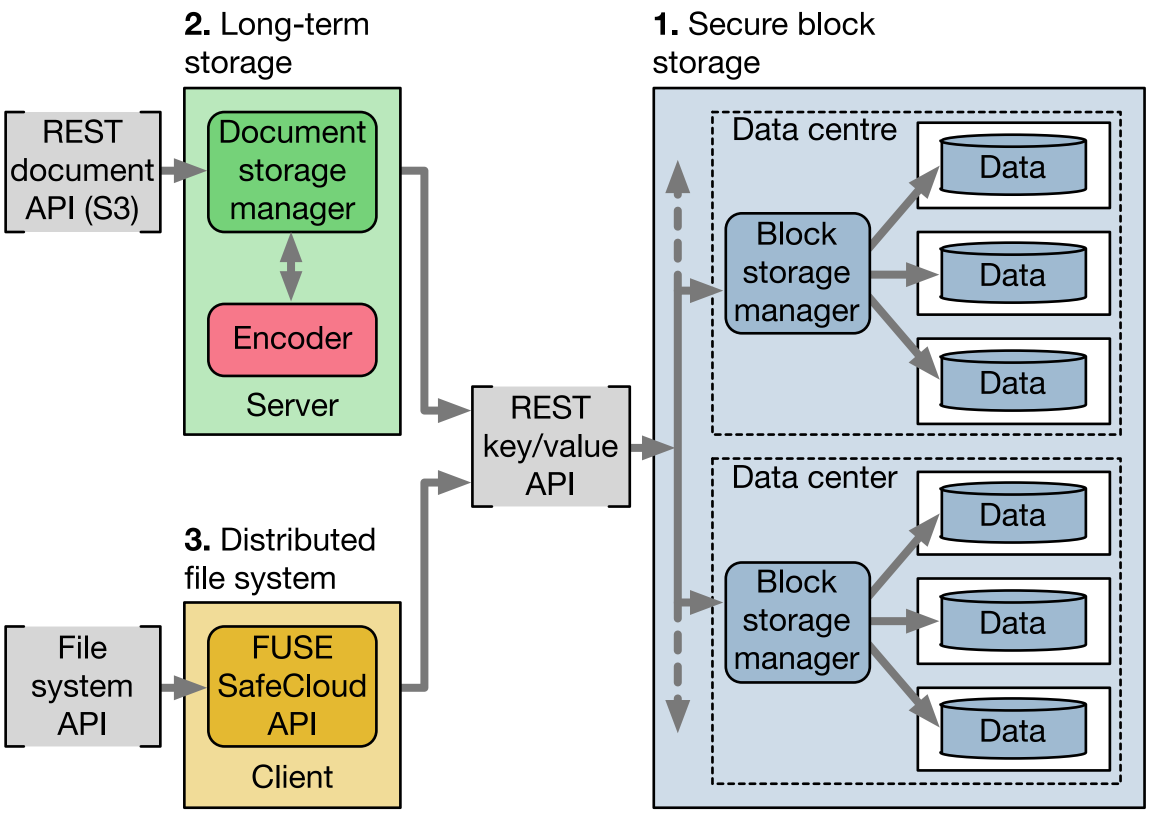


Figure 6: The three solutions of the SafeCloud storage architecture.

5.1 Solution 1: Secure block storage

The **secure block storage** solution essentially consists of a data store that can store raw data under a given key. In that respect, it behaves akin to a key/value store but provides additional mechanisms to ensure data security and integrity (using cryptographic techniques), as well as *explicit placement* of data items. As a matter of fact, being able to place different parts of data items in various geographical locations within distinct administrative domains is key to providing privacy in the SafeCloud platform. The secure block storage operates locally, on a per node basis, and there are typically several instances per data centre. Orchestration between these instances is performed by separate components, the **block storage managers**, which can explicitly place data items on individual instances of the block storage.

Placement of the data blocks can be enforced at multiple levels (e.g., administrative domain, application-defined partition, geographical region or country, data centre, rack, node, etc.) depending on some desired dependability and security properties. There can be several block storage managers at different levels of the architecture for hierarchical placement of data.

This solution provides a REST-based **key/value store API** with support for explicit placement. Such an API essentially provides operations to *store* an arbitrary piece of data under a given key that acts as a unique identifier, and to *retrieve* the data associated with a specific key. A third operation is also usually provided to *delete* a key. In its simplest form, the API thus consists of:

```
put(key, value)
value = get (key)
delete(key)
```

The API can be extended with more sophisticated operations, for instance to list or iterate through the set of keys, or to handle multiple data versions for a key. We will support the core operations initially and extend the API as needed.

5.2 Solution 2: Long-term distributed encrypted document storage

The **long-term distributed encrypted document storage**, described in more details in Section 6, builds on top of the secure block storage and supports secure storage of documents over many distributed instances of data stores. Documents are redundantly stored and protected against tampering using coding and entanglement techniques, i.e., they are encoded and combined with previous documents to ensure that no party can modify or delete them (without affecting a significant portion of all documents).

The data stored in the system is immutable, as the key idea behind entanglement is to persist documents over the long term. In other words, this solution supports archival storage. Modifications to existing data can be implemented on top of the long-term distributed encrypted document storage by the means of a versioning API, i.e., by inserting a new version of a previous document under the same name but a with different version identifier.

Clients communicate with the storage service via **document storage managers** that are responsible for splitting documents into blocks, encoding and entangling these blocks, and dispatching them onto instances of the block data store across several data centres (as well as the reverse operation for retrieving blocks and reconstructing the document). There are typically multiple document storage managers placed in different data centres and administrative domains, and clients are free to choose any of them as entry point into the system.

This solution provides a REST-based **document store API** largely compatible with the Amazon S3[1] API.

5.3 Solution 3: Distributed encrypted file system

The **distributed encrypted file system** provides a file system API on top of the secure block storage. It supports secure reading and writing of files that are geographically distributed across data centres, and hence deals with mutable data. The file system is optimized in terms of latency and throughput for sequential accesses. Placement of data in the file system can be guided by policies that express security or dependability requirements (e.g., replication degree, disaster tolerance against whole data centre failure, geo-localisation in a given set of countries, etc.).

As the file system is accessible locally from clients, it requires a local component to execute directly on the client machines. This is supported thanks to the FUSE mechanism (file system in user space).

[1] https://docs.aws.amazon.com/AmazonS3/latest/API/APIRest.html

This solution provides a FUSE-based **file system API** designed to be mostly POSIX.[2] compliant.

5.4 Integration with applications and clients

The APIs of the three solutions follow standards and are straightforward: REST-based key/value store, S3-compatible REST-based document store, and POSIX-compliant FUSE-based file system API. The actual implementation details will be addressed during the design and implementation phase of the respective tasks.

Applications that leverage these solutions use the APIs as regular clients. Depending on the solution, they will issue REST requests (for blocks or documents), or simply read/write files from/to the file system.

[2] http://standards.ieee.org/develop/wg/POSIX.html

6 SafeCloud archival using data entanglement (Solution 2 revisited)

As part of SafeCloud, we introduce STEP-archives. Using data entanglement and erasure-correcting codes, we study and develop a data storage architecture where a stored document can only be deleted or modified by compromising the integrity of other documents in the system. There are two main objectives behind this work. The first objective is data integrity. We want to provide guarantees to users that their data cannot be deleted or corrupted without compromising other data stored by themselves or other users. The second objective is to provide censorship resistance by forcing a censor who wants to tamper with data to do so noisily, and corrupt a large number of other documents in the system. An ancillary result deriving from the two objectives is increased redundancy and protection against failures, which can be seen as attacks from random or failure-specific censors.

6.1 Practical considerations

We emphasize that one of the objectives of SafeCloud is to achieve both data integrity and censorship resistance in a way implementable in practical systems. Thus, we use practical constraints that keep an actual implementation realistic (for instance avoiding reads and writes requiring a non-contant part of the total system size) while being simple enough to allow analysis. All our underlying assumptions and design choices are implementable using state-of-the-art coding and storage techniques, which is paramount for a large-scale implementation.

6.2 Entanglement architecture using erasures codes

Definition 1. A (s,t,e,p)-archive is a storage system where each archived document consists of a codeword with s source blocks, t tangled blocks, p parity blocks and that can correct e = p - s block erasures.

When a document is archived, it is split into s≥1 source blocks. Using the s source blocks with t distinct old blocks already archived, a systematic maximum distance separable (MDS) code [LC04] is used to create p≥s parity blocks which are then archived on the system.

An archived document can be recovered from s+t or more of its blocks. The code can correct p block erasures per document codeword, but since the source blocks are not archived and are considered as erased, at most $e = p - s$ block erasures per document on the storage medium can be corrected. Note that increasing t does not increase storage overhead or error-correcting capability, but does increase coding and decoding complexity.

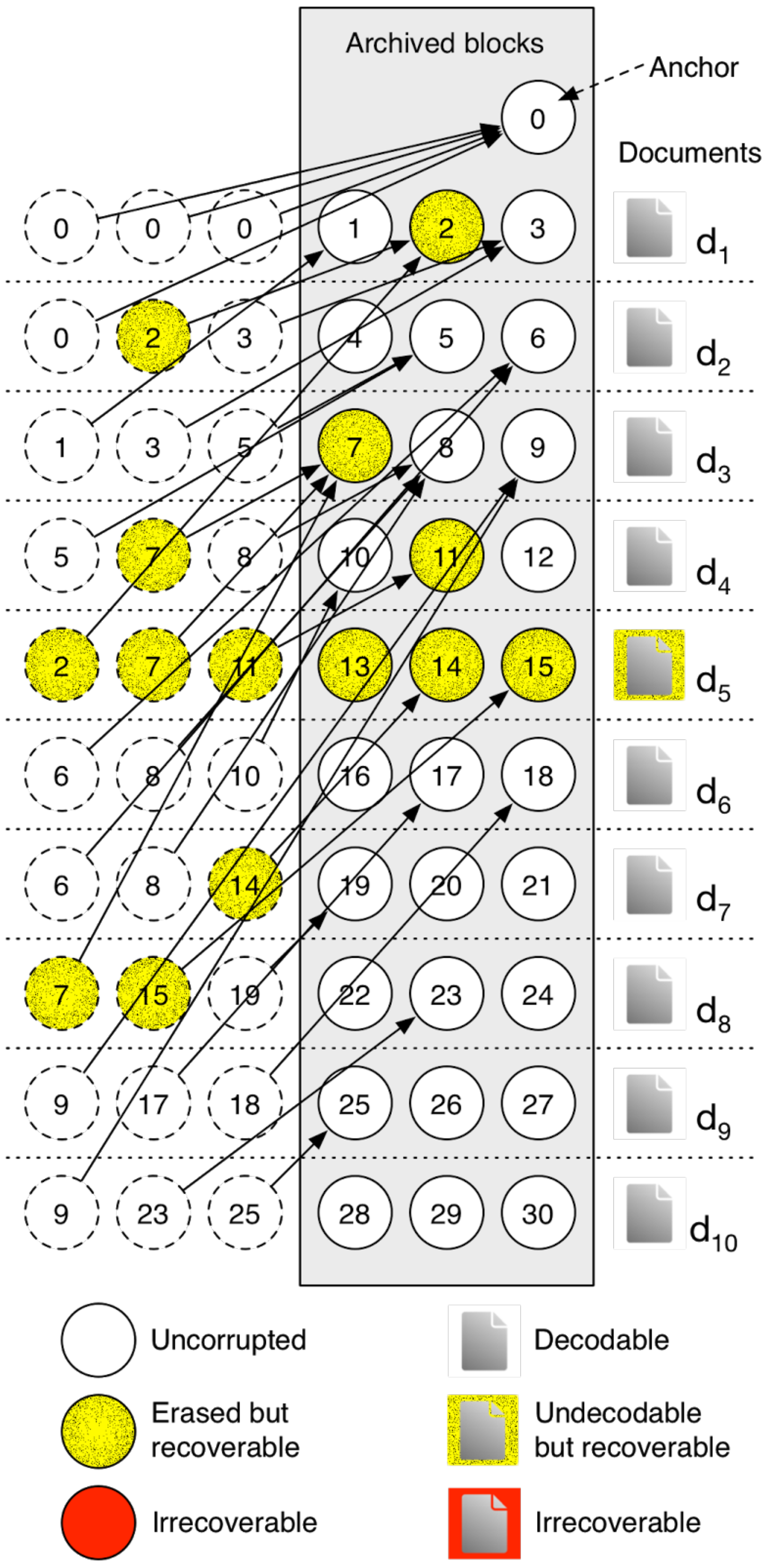


Figure 7: (1,3,2,3)-archive. 4 out of 6 blocks are required to recover a document. Recoverable attack.

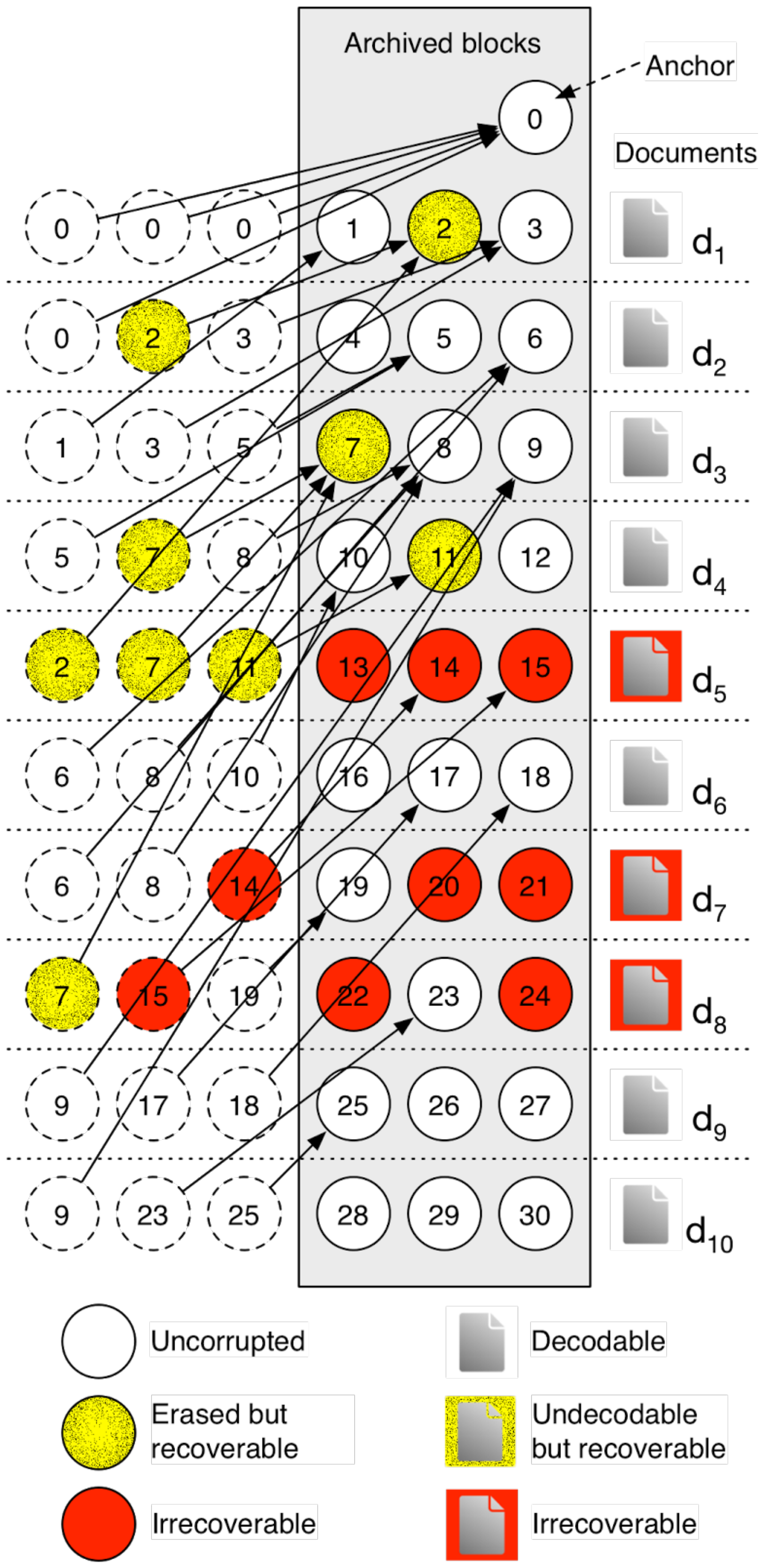


Figure 8: (1,3,2,3)-archive. 4 out of 6 blocks are required to recover a document. Irrecoverable attack.

An attacker can censor a document d_k by erasing more than e blocks from it. However, by entangling new documents with documents already archived, it might be possible for the system to recover the deleted blocks by decoding other documents that use them. As an example, consider the (1,3,2,3)-archive presented in Figure 7. Each document codeword consists of one source block, three pointers to old blocks, and three parity blocks. Only the parity blocks are stored when a new document is archived; the source block is not stored and the pointer blocks were previously stored as parity blocks of older documents. Block 0 is a known anchor that cannot be corrupted. If an MDS code is used, any four of the six stored blocks belonging to a document are sufficient to recover it (i.e., $e = 2$). In Figure 7, an attacker wants to censor document d_5 by erasing its blocks {2,7,11,13,14,15} from the archive. However, although d_5 cannot be recovered directly, all the blocks are recoverable: Block 2 can be recovered by decoding d_1 or d_2, Block 7 can be recovered by decoding d_3, $d_4$or d_8, Block 11 can be recovered by decoding d_4, Block 14 can be recovered by decoding d_7, Block 15 can be recovered by decoding d_8, and in the last step Block 13 can be recovered by decoding d_5. Having been unable to erase d_5, the attacker continues his attack more cleverly and further erases Blocks 20, 21, 22 and 24, as illustrated in Figure 8. Document $d_5$is now destroyed irrecoverably, as are also $d_7$and d_8 (the irrecoverable blocks and documents are shown in red). Blocks 2, 7 and 11 are still recoverable, which means that the attacker could have irrecoverably destroyed d_5 without destroying them.

The challenging part of our approach is thus to choose the pointers to entangled blocks in a practicable way that provides good anti-tampering and data integrity. With a non-constant number of pointers per document, maximum forward data integrity and anti-tampering is possible: on an archive with *s=1* and *e≥1*, one can use *k-1* entangled blocks for document d_k, more precisely a pointer to the first parity block of each of the *k-1* documents already archived. If a censor wants to delete a document d_k irrecoverably, it must corrupt all the documents archived after d_k. Of course, the number of pointers to tangled blocks in this example grows linearly with the number of documents, which makes encoding and decoding too complex to be of any practical value. Since we target practically implementable data integrity and anti-tampering, we thus focus on archives with t constant and small. Studying how to assign these pointers to maximise the tamper-proofing and protection against censorship will be an important task of the project.

6.3 Future investigations

We must consider and investigate many questions and problems during the project. For instance, what are the security implications of directly archiving the source blocks? Should metadata be publicly available or kept privatly? Should metadata be mass replicated to protect it against attacks? Should we consider multiple block sizes, and if so, what should those sizes be? How can we avoid a malicious administrator to practice censorship by having the main system refuse to perform read requests for censored data?

7 SafeCloud storage testbed

A core technological challenge and innovation of SafeCloud lies in its ability at using *coding techniques* to secure and entangle data in the system, as well as *place data* at specific locations guided by security and dependability requirements. To facilitate the development and evaluation of these techniques, we have designed a "storage testbed" that provides simplified versions of the SafeCloud components and operates within a single data centre, but can support identical APIs, coding techniques, fine-grained entanglement, and placement mechanisms as the final storage architecture. This testbed will not only be instrumental in driving the development of the mechanisms of work package 2, but can also be used during the implementation of the higher-level work packages until the final architecture has been completed.

The storage testbed exploits the storage capabilities of an existing key-value store to persistently store on non-volatile devices, as well as to retrieve blocks. This architecture serves as a building block to implement the more elaborate solutions, namely the long-term document store and the FUSE-based file system. In the remainder of this section we detail the various components of the architecture: a proxy service, an encoder-decoder service, and the key-value store (see Figure 9). At the time of this writing, the evaluation of various libraries and parameters that will guide future design choices is still ongoing. Results will be reported in upcoming deliverables.

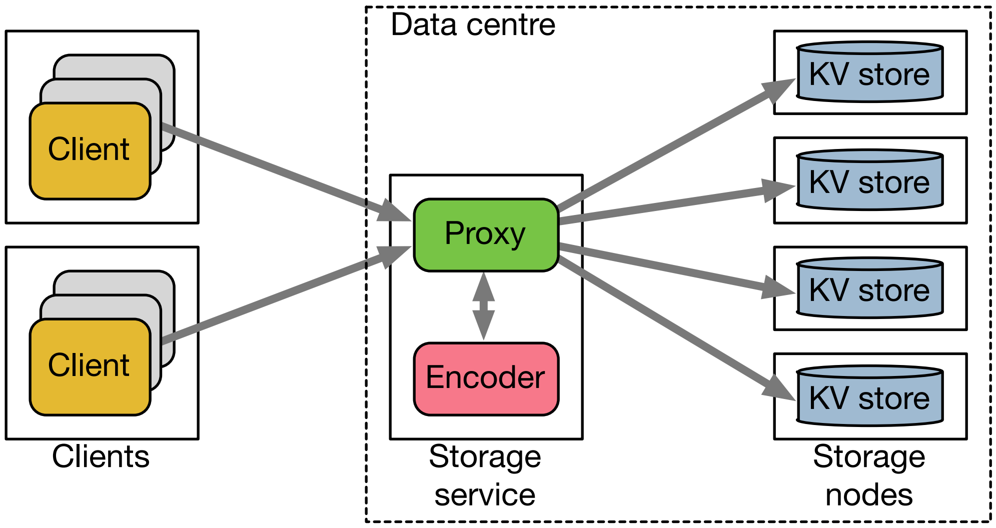


Figure 9: Storage architecture in a single data centre (testbed).

7.1 Proxy Component

The proxy component is the main front-end to the system. While there can be an arbitrary number of proxies for a given data store, we only consider one in our evaluation. The proxy exposes a simple stateless REST interface to put and get data blocks. The interface mimics the operating principles of well-established services like Amazon S3. More sophisticated operations, such as operating on subsets of the data blocks for a given file, can be easily integrated. The interactions between the proxy and the clients happen via synchronous HTTP messages over pre-established TCP channels. The proxy dispatches/collects data blocks to/from the encoder service. Note that the three solutions described in the previous section can advantageously leverage such proxies in their implementation to mediate interactions between the client and the different storage nodes across the data centres (the alternative being to embed this functionality within the client or the storage nodes).

7.2 Encoder Component

The encoder component performs the actual processing and transformation of data blocks before they are stored, as well as the reverse decoding operation. The encoder is co-located within the same host as the proxy to maximize throughput and avoid bottlenecks induced by high pressure on the network stack. To increase the flexibility of our testbed, our encoder provides a plug-in mechanism to dynamically load and swap different coding libraries. This mechanism relies on a platform-independent transport mechanism (using protobuf[3]) and a stable interface between the proxy and the encoder. The encoder interface currently exposes three main operations: encode, decode, and reconstruct.

7.3 Storage nodes

Once the blocks have been encoded, they are sent by the proxy to the storage nodes. Each storage node is independent from the others and all the interactions are mediated by the proxy. Blocks are dispatched to storage nodes using an explicit placement strategy: the proxy ensures that encoded block parts are spread to distinct nodes so that the failure of one node results in the loss of only one part. Upon reading, the proxy contacts a random minimal size subset of storage nodes to reconstruct the requested block.

7.4 Clients nodes

The clients are separate processes running on different nodes in the same data centre. They read and write data by contacting the proxy, following access patterns defined as part of the workloads.

3 https://developers.google.com/protocol-buffers/

8 Prototype implementation of storage testbed

We used different languages and technologies to implement our testbed and integrate it with the open-source erasure coding libraries. Our implementation choices have been largely driven by performance and simplicity considerations, as well as by constraints from the evaluated libraries.

The proxy component is implemented in Java and exploits the exporting facilities of the Spring Boot framework[4] (v1.3.1) to leverage industrial-grade application servers. We configured the proxy to handle HTTP requests via the embedded Jetty web-server.

The encoder component is implemented in Python to facilitate the integration with the PyEClib[5] library (v1.2), the reference Python binding for liberasure. This library implements wrappers to uniformly access several encoding libraries.

The open-source libraries under evaluation are implemented in C/C++ (e.g., liberasure,[6] JErasure,[7] LongHair[8]) or a mix of C and hand-written Assembly (e.g., Intel ISA-L[9]). These implementation choices lead to the best performance and can take advantage of hardware acceleration, as in the case of Intel ISA-L that exploits built-in SIMD CPU instructions. We call the libraries via a common access layer and software wrappers implemented in Python. Python provides an easy mean to bind to such libraries via its built-in support for native code.

The suite of macro-benchmarks leverages Apache Bench[10] (v2.3) to measure the maximum throughput and per-request latencies. The storage nodes run on top of Redis[11] (v3.0.7), a lightweight yet efficient in-memory key-value store. Redis tools offer easy-to-use probing mechanisms (e.g., the redis-cli command-line tool) to retrieve the current memory occupied by the stored keys. We exploit these tools to calculate the storage overhead of the encoded blocks.

Our deployment machinery allows us to scale all the mentioned components at will. To do so, we rely on the tools offered by the Docker[12] ecosystem (v1.6) and its Compose[13] orchestration tool (v1.5.3).

As previously mentioned, the objective of this preliminary prototype is to conduct extensive tests that will allow us to assess the performance and trade-offs of different encoding algorithms, and to consequently drive the design choices for the final SafeCloud storage architecture. Results will be reported in an upcoming deliverable as they are available.

[4] https://projects.spring.io/spring-boot/
[5] https://pypi.python.org/pypi/PyECLib
[6] https://bitbucket.org/tsg-/liberasurecode
[7] http://jerasure.org/
[8] https://github.com/catid/longhair
[9] https://github.com/01org/isa-l
[10] https://httpd.apache.org/docs/2.4/programs/ab.html
[11] http://redis.io
[12] https://www.docker.com
[13] https://docs.docker.com/compose/

9 Summary and conclusion

The objectives and contributions of the SafeCloud architecture are twofold. First, it provides a practical set of "solutions" to store content securely, at the level of raw blocks, documents, or file systems, for mutable or archival data. Second, it develops innovative techniques to advance the state of the art in data coding and entanglement. This architecture document has covered both aspects and highlighted the specific directions that will be taken to drive the SafeCloud storage infrastructure during the course of the project.

Bibliography

[ADD+12] Giuseppe Ateniese, Özgür Dagdelen, Ivan Damgård, and Daniele Venturi. Entangled encodings and data entanglement. In Feng Bao, Steven Miller, Sherman S. M. Chow, and Danfeng Yao, editors, Proceedings of the 3rd International Workshop on Security in Cloud Computing, SCC@ASIACCS '15, Singapore, Republic of Singapore, April 14, 2015, pages 3–12. ACM, 2015.

[AFY+07] James Aspnes, Joan Feigenbaum, Aleksandr Yampolskiy, and Sheng Zhong. Towards a theory of data entanglement. Theoretical Computer Science, 389(1–2), December 2007.

[And96] Ross Anderson. The eternity service. In Proceedings of Pragocrypt '96, 1996.

[APP+12] Omid Amini, David Peleg, Stéphane Pérennes, Ignasi Sau, and Saket Saurabh. On the approximability of some degree-constrained subgraph problems. Discrete Applied Mathematics, 160(12):1661–1679, 2012.

[ASS12] Omid Amini, Ignasi Sau, and Saket Saurabh. Parameterized complexity of finding small degree-constrained subgraphs. J. Discrete Algorithms, 10:70–83, 2012.

[BB89] Béla Bollobás and Graham Brightwell. Long cycles in graphs with no subgraphs of minimal degree 3. Discrete Mathematics, 75(1-3):47–53, 1989.

[BHL+08] Michael Backes, Marek Hamerlik, Alessandro Linari, Matteo Maffei, Christos Tryfonopoulos, and Gerhard Weikum. Anonymous and censorship resistant content sharing in unstructured overlays. In Proceedings of the Twenty-seventh ACM Symposium on Principles of Distributed Computing, PODC '08, pages 429–429, New York, NY, USA, 2008. ACM.

[CHL+14] Chen, H. C., Hu, Y., Lee, P. P., and Tang, Y. NCCloud: a network-coding- based storage system in a cloud-of-clouds. Transactions on Computers 63, 1 (2014), 31–44.

[CSW+01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability, pages 46–66. Springer-Verlag New York, Inc., 2001.

[DFM01] R. Dingledine, M.J. Freedman, and D. Molnar. The Free Haven Project: Distributed Anonymous Storage Service. Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000: Proceedings, 2001.

[DGW+10] Dimakis, A., Godfrey, P., Wu, Y., Wainwright, M., and Ramchandran, K. Network coding for distributed storage systems. IEEE Transactions on Infor- mation Theory 56, 9 (2010), 4539–4551.

[EFR+90] Paul Erdös, Ralph J. Faudree, Cecil C. Rousseau, and Richard H. Schelp. Subgraphs of minimal degree k. Discrete Mathematics, 85(1):53–58, 1990.

[FG06] J. Flum and M. Grohe. Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2006.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. Concrete Mathematics: A Foundation for Computer Science. Addison-Wesley Longman Publishing, second edition, 1994.

[GS99] Guruswami, V., and Sudan, M. Improved decoding of reed-solomon and algebraic-geometry codes. IEEE Transactions on Information Theory 45, 6 (1999), 1757–1767.

[GSM+03] Eu Jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. SiRiUS: Securing remote untrusted storage. In Proc. Network and Distributed Systems Security (NDSS) Symposium 2003, pages 131–145, 2003.

[CWO+11] Calder, B., Wang, J., Ogus, A., Nilakantan, N., Skjolsvold, A., McK- elvie, S., Xu, Y., Srivastav, S., Wu, J., Simitci, H., Haridas, J., Ud- daraju, C., Khatri, H., Edwards, A., Bedekar, V., Mainali, S., Abbasi, R., Agarwal, A., Haq, M. F. u., Haq, M. I. u., Bhardwaj, D., Dayanand, S., Adusumilli, A., McNett, M., Sankaran, S., Manivannan, K., and Rigas, L. Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (New York, NY, USA, 2011), SOSP ’11, ACM, pp. 143–157.

[HSX+12] Huang, C., Simitci, H., Xu, Y., Ogus, A., Calder, B., Gopalan, P., Li, J., and Yekhanin, S. Erasure coding in Windows Azure Storage. In Proceedings of the USENIX Annual Technical Conference (2012), USENIX ATC, pp. 15–26.

[JK07] Ari Juels and Burton S. Kaliski Jr. Pors: Proofs of retrievability for large files. In Proceedings of the 14th ACM conference on Computer and communications security, pages 584–597. Acm, 2007.

[KBP+12] Khan, O., Burns, R. C., Plank, J. S., Pierce, W., and Huang, C. Rethinking erasure codes for cloud file systems: minimizing i/o for recovery and degraded reads. In Proceedings of the 7th Conference on File and Storage Technologies, FAST’12, USENIX Association, p. 20.

[LC04] Shu Lin and Daniel J. Costello. Error Control Coding. Paerson Prentice Hall, second edition, 2004.

[OD15] Oggier, F. E., and Datta, A. Self-repairing codes - local repairability for cheap and fast maintenance of erasure coded data. Computing 97, 2 (2015), 171–201.

[PRW05] Ginger Perng, Michael K. Reiter, and Chenxi Wang. Censorship resistance revisited. In Proceedings of the 7th International Conference on Information Hiding, IH’05, pages 62–76, Berlin, Heidelberg, 2005. Springer-Verlag.

[PSS13] David Peleg, Ignasi Sau, and Mordechai Shalom. On approximating the d-girth of a graph. Discrete Applied Mathematics, 161(16-17):2587– 2596, 2013.

[RS60] Reed, I. S., and Solomon, G. Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics 8, 2 (1960), 300–304.

[San00] Sankaran, J. Reed Solomon decoder: TMS320C64x implementation. Application Report SPRA686, Texas Instruments, 2000.

[SGM+09] Mark W. Storer, Kevin M. Greenan, Ethan L. Miller, and Kaladhar Voruganti. POTSHARDS: a secure, recoverable, long-term archival storage system. Trans. Storage, 5(2):5:1–5:35, June 2009.

[Sha79] Adi Shamir. How to share a secret. Commun. ACM, 22(11):612–613, 1979.

[SW01] Adam Stubblefield and Dan S. Wallach. Dagster: Censorship resistant publishing without replication. Technical Report TR01-380, Rice University, July 2001.

[Tar76] Robert Endre Tarjan. Edge-disjoint spanning trees and depth-first search. Acta Informatica, 6(2):171–185, 1976.

[US12] One-way indexing for plausible deniability in censorship resistant storage. In Presented as part of the 2nd USENIX Workshop on Free and Open Communications on the Internet, Berkeley, CA, 2012. USENIX.

[Wal01] Marc Waldman. Tangler: A censorship-resistant publishing system based on document entanglements. In Proceedings of the 8th ACM Conference on Computer and Communications Security, pages 126– 135, 2001.

[WOR] Best practices to secure data from modification: Eliminating the risk to online content. GreenTec-Usa WORM White Paper. Available at http://greentec-usa.com/wp/GreenTec-WORM-Whitepaper.pdf.

[WRC00] Marc Waldman, Aviel Rubin, and Lorrie Cranor. Publius: A robust, tamper-evident, censorship-resistant web publishing system. In Proc. 9th USENIX Security Symposium, pages 59–72, 2000.

[WW08] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: the least-authority file system. In Proceedings of the 4th ACM international workshop on Storage security and survivability, pages 21–26. ACM, 2008.

[ZX12] Qingji Zheng and Shouhuai Xu. Secure and efficient proof of storage with deduplication. In Proceedings of the Second ACM Conference on Data and Application Security and Privacy, CODASPY '12, pages 1–12, New York, NY, USA, 2012. ACM.