Short-Term Secure Block Device

D2.5

Project reference no. 653884

August 2017

European Commission

Horizon 2020
European Union funding
for Research & Innovation

Document information

Scheduled delivery	31.08.2017
Actual delivery	31.08.2017
Version	1.0
Responsible Partners	UniNE

Dissemination level

Public

Revision history

Date	Editor	Status	Version	Changes
28.06.2017	R. Barbi	Draft	0.1	Initial TOC
03.07.2017	H. Mercier	Draft	0.2	Initial comments
14.07.2017	D. Burihabwa	Draft	0.3	Update evaluation
14.07.2017	V. Schiavoni	Draft	0.4	Update figs, pass on text.
14.07.2017	D. Burihabwa	Draft	0.5	Update refs
18.07.2017	H. Mercier	Draft	0.6	Small changes and comments
20.07.2017	R. Barbi	Draft	0.7	Address few comments
20.07.2017	D. Burihabwa	Draft	0.8	Address comments on source code and experiment parameters
23.08.2017	S. Schmerler	Draft	0.9	Comments re. reference list
28.08.2017	R. Barbi	Draft	0.10	Add simulation results
28.08.2017	D. Burihabwa	Draft	0.11	Pass over implementation section
29.09.2017	R.Barbi	Draft	0.12	Updating section 3
30.08.2017	V. Schiavoni	Draft	0.13	Pass on text
30.08.2017	D. Burihabwa	Draft	0.14	Update Section 4 and fix references
30.08.2017	D.Burihabwa	Draft	0.15	Update conclusion
30.08.2017	R.Barbi	Draft	0.16	Address comments
30.08.2017	D.Burihabwa	Draft	0.17	Update section 4
30.08.2017	S. Schmerler	Draft	0.18	C&H second review
30.08.2017	M. Correia	Draft	0.18	INESC-ID second review
31.08.2017	H. Mercier	Final	1.0	Final version

Contributors

R.Barbi, H. Mercier, D. Burihabwa (UniNE)

Internal reviewers

M. Correia (INESC-ID), S. Schmerler (C&H), V. Schiavoni (UniNE)

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

The deliverable presents the design of the short-term secure block device along with the description of the extended prototype implementation (Playcloud).

In D2.2, we showed how to provide anti-tampering and data integrity once data has been stored for a long period of time in the system using uniform entanglement. Unfortunately, uniform entanglement leaves recently archived documents poorly protected. In this deliverable, we discuss how to improve upon long-term entanglement to guarantee anti-tampering and data integrity to young documents. In particular, we examine the use of temporary replication together with entanglement to provide the same level of protection across the whole archive: replication protects young documents until they become old enough to be protected by entanglement.

To decrease the storage overhead of using uniform entanglement together with replication, we also study alternative heuristics for entanglement offering strong long-term protection and fast protection to young documents so that fewer of them need to be replicated.

We first discuss requirements and design considerations. We then provide details about the extended architecture of Playcloud, our experimental testbed to evaluate the performance tradeoffs of the security guarantees offered by the SafeCloud platform.

1 Introduction

The SafeCloud consortium provides three storage solutions as shown in Figure 1: secure block storage (SS1), a secure data archive (SS2), and a secure file system (SS3). This deliverable focuses on SS2. More precisely, it describes how short-term protection of archived documents is achieved. By short-term, we mean that anti-tampering and data integrity can be provided to recent data in the system. This is complementary to the description of long-term data entanglement in D2.2, and together they form the core of SS2. This deliverable also describes the extended version of Playcloud, the experimental testbed used to test all the storage solutions of SafeCloud.

This document is organized as follows. In Section 2, we summarize STEP-archives introduced in D2.2. In Section 3, we revisit how we use data entanglement in our secure data archiving system. We introduce temporary replication for short-term protection and we propose alternatives to uniform random entanglement that offer both good long-term and fast short-term anti-tampering and data integrity. Finally, in Section 4, we describe our current implementation prototype and testbed. The implementation was described in deliverable D2.2, and this current deliverable includes the progress made in the last twelve months.

Secure storage		
SS1 **Secure block storage**	**SS2** **Secure data archive**	**SS3** **Secure file system**

Figure 1: secure storage solutions.

2 SafeCloud archival using data entanglement (revisited)

As part of SafeCloud, we introduce STEP-archives, a storage system for archiving coded documents. STEP-archives were first presented in [MAL16], extensively discussed in D2.2, and further discussed in D7.10. We summarize them here for completeness. Using data entanglement and erasure-correcting codes, we 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, i.e., being forced to corrupt a large number of other documents in the system. An ancillary result deriving from the two objectives is increased protection against failures, which can be seen as attacks from random or failure-specific censors.

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 $s \geq 1$ 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 $p \geq s$ 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.

An attacker can censor a document d_k by erasing more than e of its blocks. However, by entangling new documents with documents already archived, it might be possible for the system to recover the deleted blocks by recursively decoding other documents that use them.

The challenging part of our approach is thus to choose the pointers to entangled blocks. As discussed in D2.2, in practice choosing entangled blocks uniformly at random offers three important advantages over highly structured entanglement. First, the problem is asymmetric between attackers and defenders: while a defender can efficiently recover from suboptimal attacks, an attacker must solve a NP-hard problem [APPS+12] to find a perfect (irrecoverable) attack that minimizes collateral damage, or even just approximate this minimum within a reasonable ratio. The creation of randomness in the structure prevents the attacker from planning the attack in advance, for instance by using amortized cost expensive pre-computations tied to the system structure. Second, a deterministic structure is harder to implement and maintain in real-time in a large-scale distributed setting. Third, uniform entanglement can provide strong security guarantees once data has been archived long enough.

The main drawback of uniform random entanglement is that as the archive gets bigger, it takes an increasingly longer time until new documents become properly protected. This is what we overcome in this deliverable.

3 Short-term data protection - Theory

Providing quick protection after archival is the objective of short-term data protection. The work we present in this section is twofold:

- In Section 3.1, we propose alternative heuristics to uniform entanglement that provide faster protection to recently archived documents. The time that uniform entanglement takes to protect documents is proportional to the size of the archive. To overcome this limitation, we propose normal entanglement to provide protection in a constant amount of time regardless of the size of the archive.
- In Section 3.2, we enhance the anti-censorship of recently archived documents using replication.

3.1 Entanglement heuristics

The main idea behind the alternative entanglement heuristics is to have less unprotected documents and to point to recently archived documents quickly.

In Figure 2 we show how the number of pointers per document is distributed in a (s,t,e,p)-archive with 10^5 documents. We use s=1 source block per document (not stored), t=10 pointer blocks per codeword and p=3 parity blocks per document. The erasure code offers the same fault tolerance as state-of-the-art systems, e.g. Windows Azure [HSX+12]: the erasure code can withstand 3 local erasures per codeword. This means that a document having 3 blocks not pointed to can be censored just by erasing those 3 blocks, which is highly undesirable for anti-censorship. Furthermore, the source block is not stored, thus the storage overhead on the physical medium is the same as for triple replication.

The thick black line in the figure shows the result for uniform entanglement: 10% of the documents are not pointed to, and 30% of the documents have a number of pointers insufficient to force an attacker to do collateral damage when tampering with them. This unacceptable behaviour is mainly due to the selection of the pointers from a uniform distribution over a non-constant interval: as the archive gets bigger, the number of unprotected documents increases quickly.

If every document was pointed to exactly t times, we would observe a vertical line as drawn in Figure 2. With the aim of avoiding few overly-protected and many poorly-protected documents, we propose some alternatives to uniform entanglement:

- Normal entanglement (N-std1000): we select pointers following a normal distribution having as mean the index of the document we want to store and standard deviation σ=1000. More precisely, as we can only point to archived documents, this corresponds to the left half of a normal distribution with standard deviation σ=1000.
- Mix of normal and uniform entanglement: in rand5+norm and rand6+norm we use respectively 5 and 6 random pointers from the complete pool of archived documents and the remaining pointers following a normal distribution with standard deviation σ=100.
- Mix of uniform entanglement (rand+rand): 6 pointers are chosen at random within a sliding window of size 50 and 4 pointers are chosen at random from the whole pool of blocks.

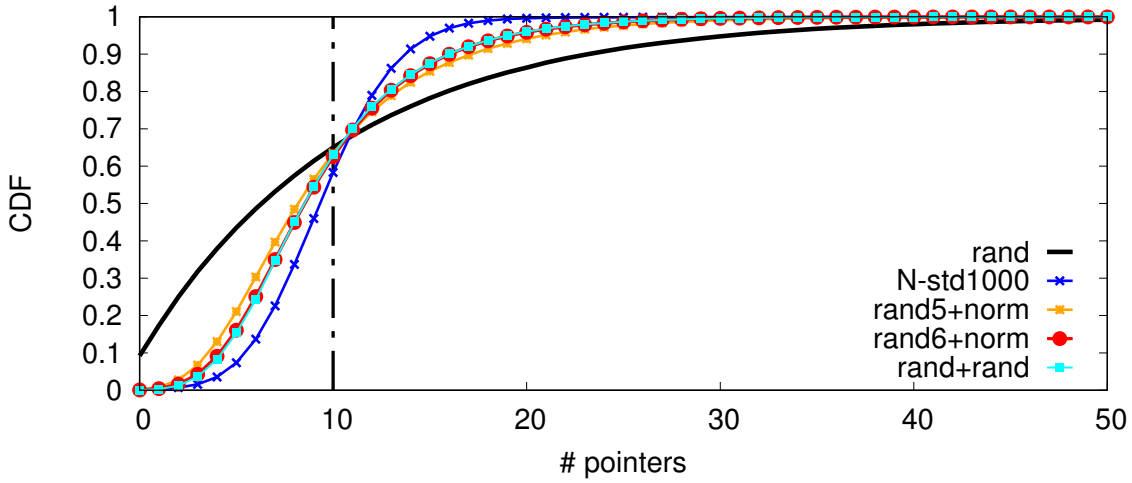

Figure 2: Cumulative Distribution Function (CDF) of the number of pointers per document for an archive of 100000 documents. The configuration of the (s,t,e,p)-archive is s = 1, t = 10, p = 3.

As Figure 2 illustrates, all these strategies reduce the number of document poorly protected. In particular, the heuristic N-std1000 enables documents to get protection faster than the other strategies and reduces the number of documents that can be censored with no collateral damage.

We thus focus on normal entanglement because it provides good long-term protection, fast short-term protection, and a level of replication independent on the size of the archive as we show in the remaining of the section.

3.2 Replication

Regardless of the heuristic to select pointers, entanglement takes time to provide protection to new documents (for instance the last archived document is never pointed to). We thus use temporary replication to guarantee the same level of protection across the whole archive. Spread over the various storage nodes, the replicas ensure a fast recovery in case of failure of a node but significantly add to the storage overhead.

In order to prevent the explosion in storage cost, we periodically examine the level of protection of blocks and once a given threshold is passed, remove their replicas from the system. In the following we study how to set such a threshold. We evaluate by means of simulations when a block's replicas can be safely removed.

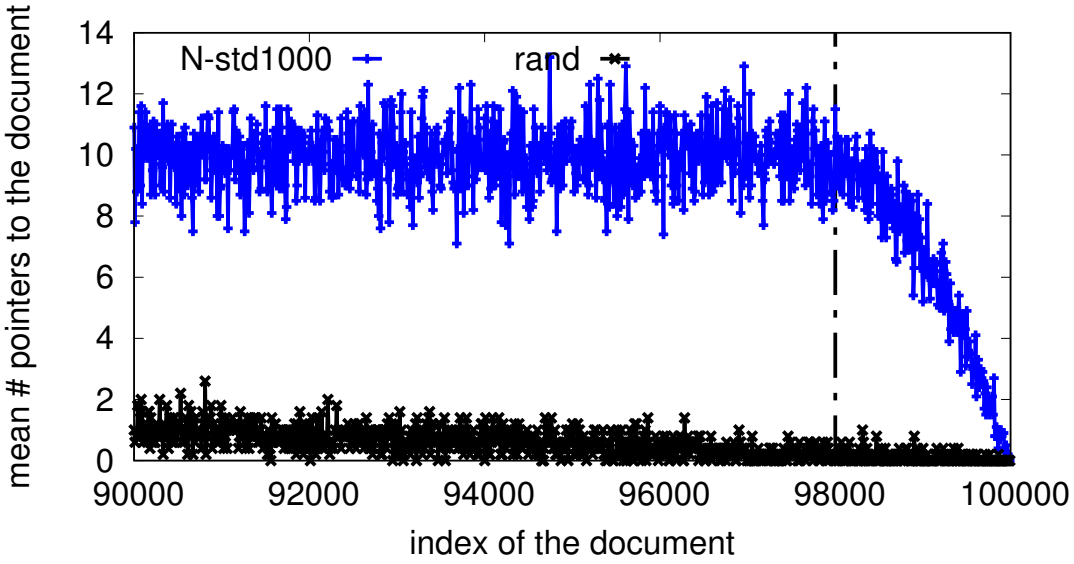

Figure 3: Distribution of the number of pointers (mean over 10 simulations) to each archived document in an archive with 100000 documents. The configuration of the (s,t,e,p)-archive is s = 1, t = 10, p = 3.

For uniform entanglement, as the number of blocks increases over time, the random selection of pointers lowers the probability of picking recent ones and in consequence raises the lifetime of their replicas. In brief, the number of documents that must be replicated is proportional to the number of documents stored. Figure 3 shows clearly that at least 10% of the documents needs to be replicated, indeed the document with index 90000 is pointed to, on average, only by 2 other documents. On the other hand, using normal entanglement with standard deviation σ=1000, only the last 2σ=2000 documents are poorly protected. This number does not depend on the size of the archive but only on the standard deviation σ. This is illustrated in Figure 3.

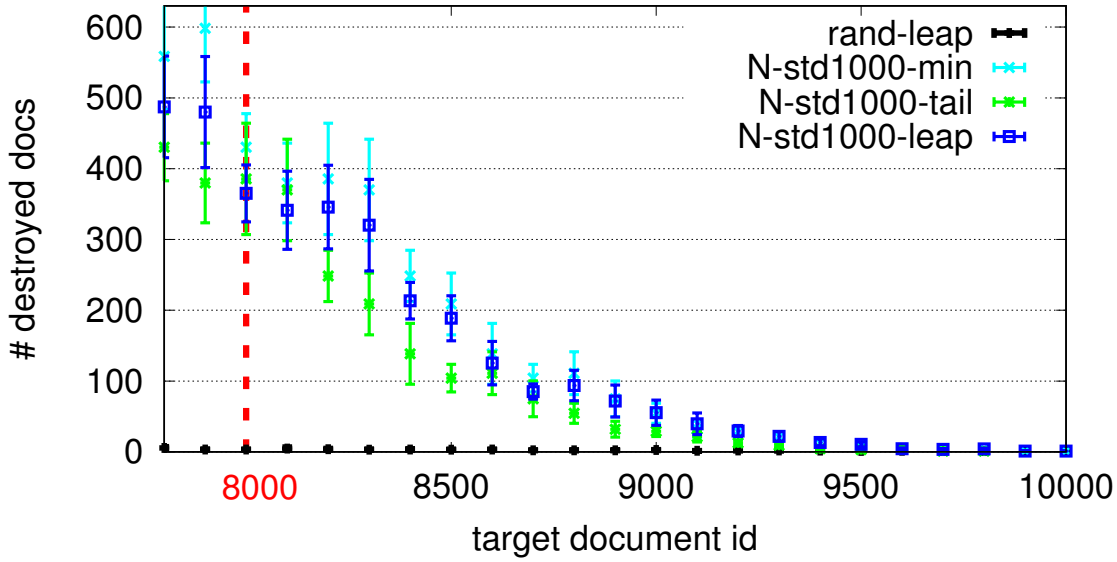

Figure 4: Mean and interquartile range for the number of documents corrupted by different attacks to censor the x-axis-indexed document. The attacks are the leaping attack (random and normal entanglement), the minimum attack (normal entanglement), and the tailored attack (normal entanglement). The attacks are described in [MAL16] and in D2.2. The configuration of the (s,t,e,p)-archive is s = 1, t = 10, p = 3.

To confirm this intuition, we archive 10000 documents and run the greedy attacks proposed in [MAL16] and presented in D2.2, with the aim of censoring recently archived documents, i.e., documents with index i such that $7800 \leq i \leq 10000$. We show the results in Figure 4. As expected very young documents can be censored by tampering with a few documents. However, with normal entanglement the protection is fast, and to erase document 8000 the attacker needs to tamper with more than 300 documents. We recall that finding the minimum number of documents to censor a target document is a NP-hard problem [MAL16] and, as a consequence, the greedy attacks are sub-optimal heuristics proving an upper bound. On the other hand, using uniform entanglement document 8000 can be censored by tampering with 2.3 documents on average. We highlight that in Figures 3 and 4, the number of documents in the archive is 10^5 and 10^4, respectively. Nevertheless, the threshold between protected and unprotected documents is $2\sigma=2000$ documents before the end of the archive. As discussed above, when using normal entanglement, the threshold is determined by the standard deviation σ of the normal distribution and does not depend on the size of the archive.

The last set of simulations we run is dedicated to check that switching from uniform to normal entanglement does not ease the work of a censor targeting an old enough document.

The heuristic N-std1000 defines a sort of sliding window whose width is determined by the standard deviation σ of the normal distribution: intuitively, introducing a sliding window might facilitate the job of the censor as it decreases the space where pointers are spread. We run the greedy attacks from [MAL16] explained in Section 4.2 of D2.2 to check that the standard deviation $\sigma=1000$ is big enough for preventing the censor to exploit the sliding window. In particular, we want to force an attacker tampering with any target document in the archive to recursively spread the attack to the largest possible number of documents.

We evaluate how normal entanglement affects the number of documents to be corrupted to censor one target document and compare it against uniform entanglement. We report the results in Figure 5. This figure should be read as “the higher the better” indeed the purpose of entanglement is to force the attacker wishing to censor a document to do it noisily affecting a large part of the archive.

In Figure 5, the heuristics causing the least damage to censor the target document are the leaping and tailored attacks. For instance, when 5000 documents have been archived, the fraction of documents to be erased by the leaping attack to censor d_1 is 0.77 for uniform entanglement and 0.74 for normal entanglement, so it is slightly easier to censor the first document archived if we use normal entanglement. This difference is not significant because a large fraction of documents must be corrupted in both cases. Much more important is the significant advantage of normal entanglement over uniform entanglement for newly archived documents, as discussed above.

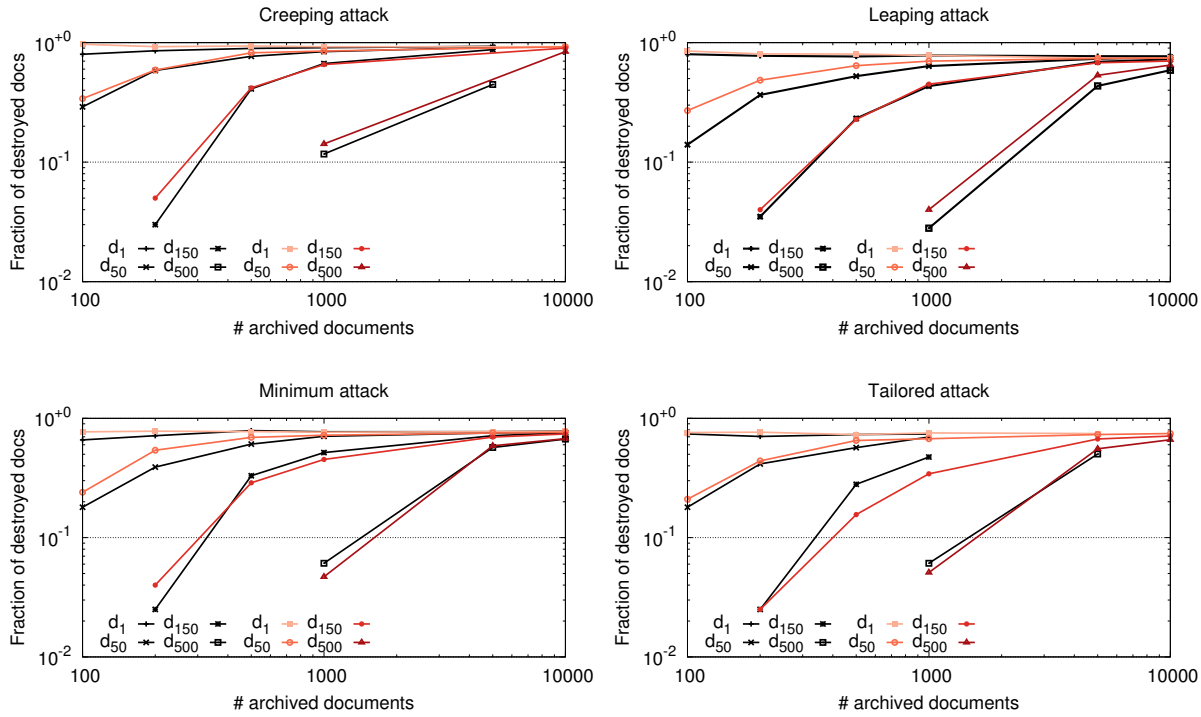

Figure 5: Fraction of corrupted documents required to erase the target document {d_1, d_{50}, d_{150}, d_{500}} indicated in the legend. On each graph, the two curves for each target document compare uniform entanglement (black curves) and normal entanglement (colour curves). The configuration of the (s,t,e,p)-archive is s=1, t=10, p=3.

Summarizing, uniform entanglement provides strong long-term protection but needs a massive use of replication to reach an adequate level of short-term protection.

To reduce the storage overhead, we propose normal entanglement which provides comparable long-term protection, fast short-term protection and a level of replication independent on the size of the archive.

We stress that while the cost of replicas grows linearly with the size of the archive if we use uniform entanglement, it goes to zero in the case of normal entanglement. Indeed, as the number of documents that need to be replicated is constant, in the long term, it becomes a negligible fraction of the whole archive.

4 Short-term data protection - Implementation

To evaluate the performance impact of the short-term data protection, we implement the scheme in Playcloud an experimental test bed previously introduced in D2.2 and extensively evaluated in [BPF+16] (still designated by its original name SafeStore). The source code of Playcloud is currently in a private repository, and might be released publicly as it reaches a more mature state.

This section briefly covers the architecture of Playcloud, the changes introduced by short-term protection and their impact on performance.

4.1 Architecture

The Playcloud architecture comprises the following components: a storage server ("proxy") that mediates interactions between clients and the Playcloud system, an encoder component, and a set of backend storage clouds (public clouds or private servers deployed on-premises). Figure 6 presents an instance of Playcloud connected to a set of clients and various cloud storage providers.

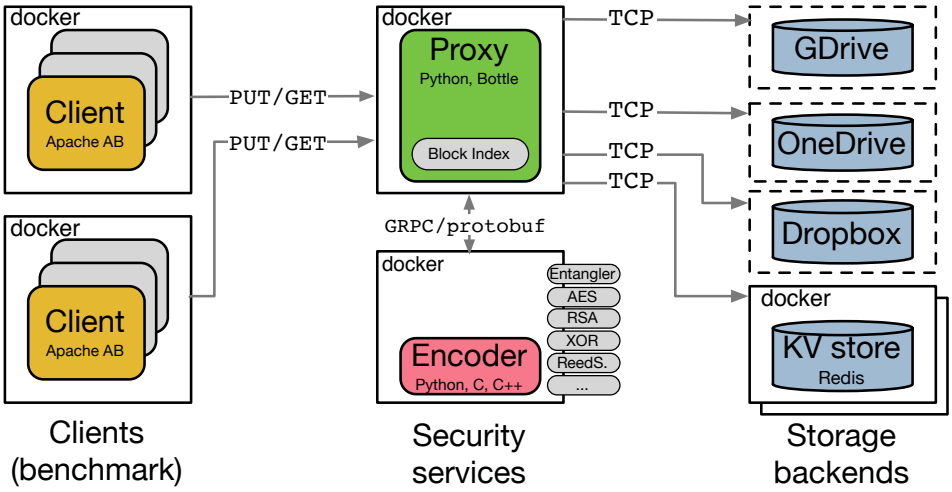

Figure 6: Architecture of our experimental testbed.

The proxy component acts as the Playcloud's front-end and is responsible for keeping a mapping between client's files and the actual storage backends where these are stored. Clients, which run in independent nodes, contact the proxy component to write or read data through a simple REST interface that mimics the operating principles of well-established services like Amazon S3. The interactions between the proxy and the clients happen via synchronous HTTP messages over pre-established TCP channels. Serving as the entrypoint to the system and holder of the metadata, it is essential that the proxy be trusted and advisable to have it deployed client-side.

The Playcloud system is configurable and different security mechanisms can be put in place. According to such configuration, the proxy component coordinates the other components in the system and different workflows may arise. For instance, some configurations require a single cloud backend while others require two or more. Upon a write request, the proxy component asks the encoder component to encode data blocks according to the configured security mechanisms. The resulting block or blocks are then dispatched by the proxy to the storage backends. To this end, the proxy maintains a data block index to keep track of where data is stored at the backends. The proxy also takes care of replication when inserting a new document. Additionally, and for the case where anti-censorship mechanisms are in place, the encoder maintains an entangler component.

Upon a read request for a piece of data, the proxy checks the block index to figure out where the corresponding encoded blocks are stored. It fetches them from the backend storage and forwards them to the encoder that decodes the blocks before returning the data to the client. The encoder is co-localized 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 plugin mechanism to dynamically load and swap different coding and cryptographic libraries and associated bindings. This mechanism relies on a platform-independent transport mechanism (using protocol buffers) and a stable interface between the proxy and the encoder.

4.2 The Playcloud API

Once started, a Playcloud instance can be interacted with using an HTTP client such as cURL [curl] by sending PUT and GET requests. The PUT command is used to insert documents into Playcloud while the GET command is used to retrieve those documents from the system.

A user willing to store a document can issue the following command to store a new document named `report.pdf` into the system.

```
curl –X PUT http://<server>:<port>/ -T report.pdf
```

Retrieving the newly inserted document can be done by running the following command.

```
curl –X GET http://<server>:<port>/report.pdf -o copy.pdf
```

These commands should run transparently for the user regardless of the internal configuration chosen by the administrator.

4.3 Entanglement and replicas management

Besides the simple exclusive-or-based entanglement approach based on Dagster [SW01] previously implemented and described in D2.2, we implement the (s,t,e,p)-archive to provide anti-censorship. In combination with short-term data protection, these changes modify the way data is stored and managed over time in the system. In short, the lifecycle of a document and the blocks that it is made of can be described as follows:

1. We perform uniform random entanglement as described in Section 3;
2. We compute a fixed number of replicas and spread them randomly;
3. We check if blocks pointed to by the newly inserted document have reached a given level of protection and, if they do, delete their replicas;

This last change introduces two new parameters to configure the system: the replication factor and the protection threshold.

The first parameter, the replication factor, is known by the proxy as it uses this number to compute how many copies of any block it must store on its data nodes. The coder, unaware of the replication factor, only asks for blocks and never a specific replica on a specific storage node. This makes the management of replicas easier and shields the coder or any other client from having to deal with corrupted replicas or unavailable storage nodes. As a bonus, in a setting with homogenous storage nodes, this level of indirection enables the proxy server to balance the load of read requests for a block over the different storage nodes holding replicas.

The second parameter, the protection threshold, is also known and used by the proxy. In the current prototype implementation, the threshold is expressed as the number of documents pointing at a block. The choice of this metric rather than the age of the document and its blocks stems from our use of uniformly random selection of pointers.

As described in Section 3, some old blocks may never be entangled with enough documents to reach the given threshold which prevents the system to use an age-based approach. In practice, a separate process running in the proxy component, periodically scans the metadata looking for the blocks that have been pointed at a given number of times and once this list has been assembled, removes the redundant replicas and updates the proxy's metadata.

While tracking the pointer count to a block and deleting accordingly provides a first step in preventing the storage overhead from becoming prohibitively costly, the observations made in Section 3 as well as preliminary simulations confirm the idea that implementing pointer selection using a normal distribution should be favoured. Work on the integration of pointer selection using a normal distribution is currently ongoing and will be presented in D2.7.

4.4 Implementation Details

Our implementation choices have been largely driven by performance and programming simplicity considerations, as well as by constraints from the storage backend interfaces.

The proxy component is implemented in Python (v2.7.11) and exploits the exporting facilities of the Bottle [bot] framework (v0.12.9). The proxy handles PUT and GET requests via the WSGI [wsgi] Web framework.

The encoder, also written in Python, integrates with various encoding libraries. Each library is wrapped exposing the same API to the encoder allowing the system to be expanded and to abstract Playcloud from the implementation details of each library. This allows Playcloud to support not only Python libraries but also native ones.

As the erasure coding driver, Playcloud supports Jerasure, an efficient Cauchy Reed-Solomon driver implemented in C/C++ that is exported by the PyEClib [pye] library (v1.2), while the entanglement component is implemented in Python.

For the client side, we built a suite of micro- and macro-benchmarks, leveraging the Yahoo Cloud Serving Benchmark (YCSB) [ycsb] and Apache Bench [ab], to measure the throughput and latency of client storage requests.

Finally, we have implemented drivers for four storage backends. First, we deployed a set of on-premises storage nodes using Redis [redis] (v3.2.8), a lightweight yet efficient in-memory key-value store. Redis tools provide easy-to-use probing mechanisms (e.g., the redis-cli command-line tool), which allowed us to measure the impact of the several security combinations used in our evaluation. Second, we have implemented drivers for the three most widely used cloud storage services: Dropbox [dbox], Google Drive [gdrive], and Microsoft OneDrive [odrive]. The drivers are implemented leveraging the official Python SDKs from each provider. Similarly to the approach taken with the encoding component, storage backends are wrapped to expose a common interface with the required set of operations, i.e., store, fetch and delete data, which allows to easily plug-in new storage backends in the future. Overall, our implementation consists of 10872 lines of Python code, all components included.

4.5 Evaluation

This section presents our evaluation of the performance impact of the short-term data protection scheme.

4.5.1 Evaluation Settings

We deploy our experiment over a cluster of machines interconnected by a 1 Gb/s switched network. Each physical host features 8-Core Xeon CPUs and 8 GB of RAM. We deploy virtual machines (VM) on top of the hosts. The KVM hypervisor, which controls the execution of the VM, is configured to expose the physical CPU to the guest VM and Docker containers by mean of the host-passthrough [kvm] option, to allow the encoders to exploit special CPU instructions. The VMs leverage the Virtio module for better I/O performances. The different components are deployed over the VMs as Docker[docker] (version 17.06.0-ce) containers.

We deploy the components over 18 VMs as follows:

- 1 VM for the proxy container and the encoder container;
- 1 VM for the client;
- 16 VMs each hosting a Redis container.

4.5.2 Impact of short term data protection on request response time

The figure below presents the response times of 500 requests to store a document of 1MB into our system. The different curves displayed illustrate the changes in performance as the entanglement scheme or the number of replicas stored are modified.
The observations from these experiments are two-fold. First, the STeP scheme is noticeably slower than Dagster. Second, the introduction of replication does not incur a significant overhead for the system (2.13% for Dagster and 3.15% for STeP). We explain these results by the complexity of the STeP scheme that requires the same number of blocks for entanglement but encodes them in a different way and, in return, produces two extra blocks that must be stored. In practice, the overhead in storage and latency is balanced by the fact that STeP provides redundancy guarantees out of the box while Dagster relies on replication to be able to recover from erasures. Furthermore, Dasgter has been proven to offer a lower level of censorship resistance in [AFYZ07] as the average number of documents affected by the destruction of a block is not high enough to serve as a deterrent for the attacker.

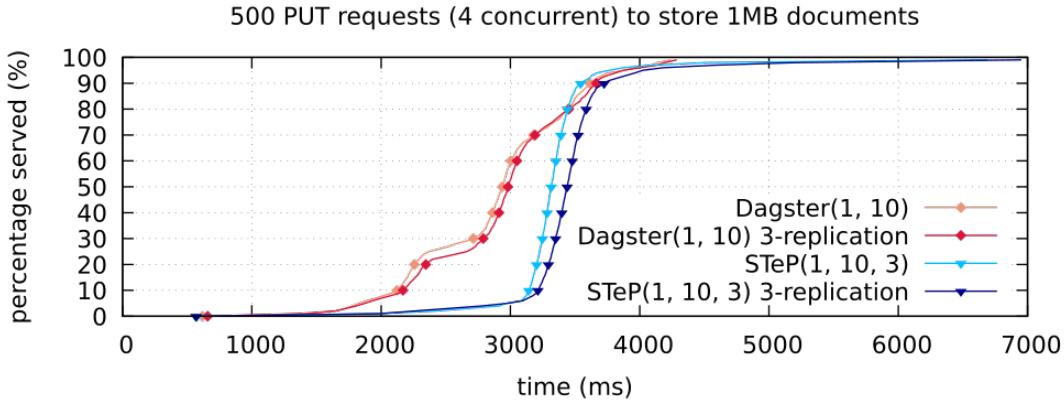

Figure 7: CDF of response times for requests to store 1MB documents in Playcloud using various entanglement configuration storing 1 or 3 replicas of the blocks on the storage nodes.

5 Conclusion

This deliverable presents the short-term secure block device, which extends and enriches the secure data archive (SS2) presented in D2.2 by enforcing the same level of anti-tampering and data integrity in the whole STEP-archive. The use of replication to protect younger documents paired with a smarter pointer selection for entanglement using a normal distribution offers both long-term and short-term protection. By simulation and experimentation on our prototype implementation, we show that short-term replication can be achieved at a reasonable cost in terms of performance and storage overhead.

Playcloud will be further extended and optimized in D2.7 to provide better pointer selection, block placement and replication management strategies as new theoretical results emerge.

6 References

[ab] Apache Bench, https://httpd.apache.org/docs/2.4/programs/ab.html.

[AFYZ07] J. Aspnes, J. Feigenbaum, A. Yampolskiy, and S. Zhong, Towards a theory of data entanglement, Theoretical Computer Science, vol. 389, no. 1–2, Dec. 2007.

[APPS+12] Omid Amini, David Peleg, Stéphane Pérennes, Ignasi Sau and Saket Saurabh, On the approximability of some degree-constrained subgraph problems, 2nd USENIX Workshop on Free and Open Communications on the Internet, 2012.

[bot] Bottle, https://bottlepy.org/docs/dev/.

[BPF+16] Dorian Burihabwa, Rogerio Pontes, Pascal Felber, Francisco Maia, Hugues Mercier, Rui Oliveira, Joao Paulo and Valerio Schiavoni, On the Cost of Safe Storage for Public Clouds: an Experimental Evaluation, , 2016.

[dbox] Dropbox, https://www.dropbox.com/.

[docker] Docker, https://www.docker.com/.

[gdrive] GDrive, https://www.google.com/drive/.

[kvm] http://www.linux-kvm.org/page/Tuning_KVM.

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

[MAL16] Hugues Mercier, Maxime Augier and Arjen K. Lenstra, STEP-archival: Storage Integrity and Tamper Resistance using Data Entanglement, Submitted to the IEEE Transactions on Information Theory, 2016 (revised 2017). Available upon request.

[odrive] OneDrive, https://onedrive.live.com/.

[pye] PyECLib, https://github.com/openstack/pyeclib.

[redis] Redis, https://redis.io/.

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

[wsgi] WSGI, http://www.wsgi.org.

[ycsb] YCSB, https://github.com/brianfrankcooper/YCSB.