STEP-Archival: Storage Integrity and Tamper Resistance Using Data Entanglement

Hugues Mercier, Maxime Augier, and Arjen K. Lenstra

***Abstract*— We present STEP-archives, a novel and practical data archival architecture, where an attacker who wants to censor or tamper with a data object must cause obvious collateral damage to a large number of other objects in the system. We use maximum distance separable erasure codes to entangle unrelated data blocks and provide redundancy against storage failures, which results in an archive with constant time read–write operations. We show a tradeoff for the attacker between attack complexity, irrecoverability, and collateral damage. We also show that the problem is asymmetric between attackers and defenders; while a defender can efficiently recover from imperfect attacks, an attacker must solve an NP-hard problem to find a perfect (irrecoverable) attack that minimizes collateral damage to other data objects, or even approximate its size. We then study efficient sample-heuristic attack algorithms that lead to irrecoverable but large damage and demonstrate how some strategies and parameter choices allow to resist these sample attacks. Finally, we provide empirical evidence that an attacker who wants to irrecoverably tamper with a document archived long enough must destroy a constant fraction of the archive.**

***Index Terms*— Distributed storage, data archival, anti-tampering, anti-censorship, data integrity, data entanglement, MDS codes.**

I. Introduction

THE way we store and archive data is being transformed by the emergence of information and communication technologies. Among these technologies, distributed file storage, sharing and synchronization systems on the cloud are becoming ubiquitous, and all major information technology players are offering some flavor of it: Dropbox, iCloud, Google Drive, OneDrive, Amazon S3, ... They provide easy access to data from multiple devices and locations as well as protection against data loss from hardware failures. However, recent developments in the wake of expansive and sometimes unauthorized government access to private and sensitive data raise major privacy and security concerns about data located in the cloud, especially when data is physically located or transits outside the legal jurisdiction of its rightful owner.

Manuscript received July 18, 2016; revised August 19, 2017; accepted March 21, 2018. Date of publication April 12, 2018; date of current version May 18, 2018. This work was supported in part by the Swiss National Science Foundation Sinergia on Trustworthy Cloud Storage under Project 136318 and in part by the European Union's Horizon 2020–The EU Framework Programme for Research and Innovation 2014-2020 under Grant 653884. This paper was presented at the 2015 International Symposium on Information Theory.

H. Mercier is with the Institutes of Computer Science and Mathematics, Université de Neuchâtel, 2000 Neuchâtel, Switzerland (e-mail: hugues.mercier@unine.ch).

M. Augier was with the IC LACAL, EPFL, 1015 Lausanne, Switzerland. He is currently an independent researcher.

A. K. Lenstra is with the IC LACAL, EPFL, 1015 Lausanne, Switzerland.

Communicated by Alfred Menenzes, Associate Editor for Complexity and Cryptography.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2018.2825981

In this article, 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 and will likely remain properly, securely, and reliably archived. 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 his data will be as securely and reliably stored as data from more important customers using the same service? These problems are especially relevant with old archives, some of which might not need to be accessed for decades. Despite remarkable recent theoretical and technical progress, users often still rely in the good faith they have in their providers (and in the catastrophic consequences for their providers' bottom line should they lose the data).

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 sensitive information that can be viewed as threatening or compromising by potential censors, with incentives worth billions of dollars to tamper with it. A good archival system must thus make it 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 exist plenty of solutions from the client perspective using client-side cryptography.

There is currently no practical software-based archival system providing strong anti-tampering and anti-censorship. Designing such a system is a difficult endeavor, both in theory and in practice. This is the main objective of our work.

A. State of the Art

1) Censorship-Resistant File Sharing: Censorship-resistant file sharing systems and overlays have been studied in various forms and for a large number of settings and applications, starting with the Eternity Service proposal [1]. This was followed by Publius [2], Freenet [3], Free Haven [4], Dagster [5], Tangler [6], SiRiUS [7], Tahoe [8], and Clouds [9]. All these solutions are summarized and put in context in a recent survey on privacy and decentralization [10].

Randomized encryption can prevent a malicious storage system from extracting information about its users, by observing which users independently store identical files (specifically, it prevents the system from noticing when unrelated files are identical). It can also be used to prevent the system from preemptively censoring documents corresponding to known content. This has been implemented with success in practice, notably by the Tahoe [8] filesystem. 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.

Perng *et al.* [11] point out that none of the above solutions quantifies the amount nor the quality of anti-censorship they provide, reverting instead to statements such as "Our system should make it extremely difficult for a third party to make changes to or force the deletion of published materials" [2]. They also show that the censorship-resistance of these systems is not comprehensive. To overcome this lack of formalism, [11] provides the first definition of censorship resistance by modeling the adversary as a selective filter that must block a targeted document while allowing the retrieval of the other documents. They show that private information retrieval [12] is a necessary but insufficient condition to guarantee censorship resistance in this context.

2) Data Entanglement: The idea behind data entanglement is that an adversary who wants to censor or tamper with a specific piece of data must also censor or tamper with tangled data elsewhere in the system. Providing anti-censorship using data entanglement was first proposed by Dagster [5] and Tangler [6]. 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, which on average will destroy $O(c)$ 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 [13]. The two new blocks are then stored. The original document can be recovered with any three of the four blocks using Lagrange interpolation. In [14], it is shown that erasing two blocks from a random Tangler document erases on average $\mathcal{O}(\frac{\log n}{n})$ 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.

A theory of data entanglement is developed in [14]. In this setting, clients submit their files to a trusted third party that entangles them into a common store. Clients who want to retrieve their file use recovery algorithms. An important contribution is the introduction of all-or-nothing integrity: intuitively, either all the documents are recoverable with high probability, or no document is. The authors show that all-or-nothing integrity is possible with some restrictions on the power of the attacker. Ateniese *et al.* [15] extend the work by allowing stronger all-or-nothing integrity without a trusted third party. They also present a simulation-based security analysis in the universal composability model, and guarantee the privacy of the data and randomness used to encode it in the presence of an adversary in possession of a subset of this information. Unfortunately, the protocols provided in both articles [14], [15] remain far from real-life implementations: users must entangle their data in a single, costly and coordinated fashion, and they cannot add new data without re-entangling everything from scratch. Furthermore, the scheme is based on polynomial interpolation whose degree depends on the number of users. As importantly, no data is recoverable if the storage provider corrupts or fails to maintain a small part of the data.

3) WORM Storage: For compliance and legal reasons (such as SEC Rule 17a-4 [16], [17]), sensitive data may be stored using a "write-once, ready-many" (WORM) technology. WORM storage is a niche market of secure data storage solutions that has been historically fulfilled using hardware approaches. It 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, ... Physical implementations typically offer more constrained data access than logical approaches and are more dependent on hardware robustness against failures and destruction. To alleviate these shortcomings, recent solutions integrate WORM hardware with software control, where the protection against data rewrite is embedded at the physical disk level. There is little peer-reviewed work on WORM technologies [18], and parties interested in the state of the art must consult recent business white papers and patents from providers such as Dell EMC [19], GreenTec [20], HP [21] HubStor [22] and IBM [23], [24]. As pointed out in [18], WORM technologies seek compliance as much as security, and do not provide strong WORM security such as guaranteed retention despite insider's physical access.

4) Proofs of Storage: A significant amount of work on provable data possession [25] and proofs of retrievability [26] has emerged in the last ten years. The idea behind provable data possession is that clients sign data prior to transmitting it to a remote server. Once data is stored, clients can audit the remote server by challenging it to prove that randomly chosen data blocks are still in its possession. The answer takes the form of a constant size proof of possession, and with high probability the server will be unable to answer correctly if the blocks are corrupted. Proofs of retrievability use error-correcting codes on top of signed data. This allows the client to challenge the remote server to succinctly prove that it has the entire data object in its possession with high probability. Work on these topics was extended to tackle many issues such as proof of multiple stored replicas [27], scalability [28], deduplication [29] and data updates [30]. All these mechanisms allow a client to learn whether a remote server still has the data at a specific time when challenged. They cannot guarantee that the server will agree to disclose the data when prompted to do so and thus offer no censorship protection, nor do they protect against tampering.

B. Our Contributions

In this article, we introduce STEP-archives. Using data entanglement and erasure-correcting codes, we propose, develop and analyze a novel 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 tamper and censorship resistance by forcing an adversary who wants to censor or 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.

1) Attacker - Defender Asymmetry: An attacker who wants to tamper with a document must try to destroy it irrecoverably by recursively eliminating all cascading dependencies in other documents. One of the interesting aspects of our approach is its asymmetry. On the one hand, it is easy to repair the system if the damage done by an attacker is recoverable. On the other hand, we prove that irrecoverably destroying a target document while minimizing the number of other documents destroyed by the attack is **NP**-hard. Even approximating the size of the smallest irrecoverable attack is hard, unless $\mathbf{P} = \mathbf{NP}$.

2) System Robustness and Forward Integrity: In the data entanglement work of [14] and [15], all-or-nothing integrity makes it essentially impossible to recover anything if the storage provider corrupts or fails to maintain a small part of the data. Although this feature is a strong incentive for the storage provider to behave responsibly, it has a perverse and undesirable effect: a malicious attacker, having compromised an honest provider, can deny access to all the data by denying access to a small part of the system, and irrecoverably destroy the entire data by corrupting a small amount of it.

We take the dual approach to achieve the same objective and introduce the concept of forward integrity. We use an append-only coding scheme for which unrelated pieces of data eventually become mutually dependent. When a document has been archived long enough, it can only be lost by destroying a large number of other documents. Furthermore, for an archive to reach the state where old documents are irrecoverably destroyed, the storage provider must be very sloppy, or the attacker must be very powerful and willing to do a lot of work.

3) Entanglement Strategies and Suboptimal Attacks: After introducing and describing our architecture, we present different entanglement strategies and suboptimal attacks within this model, and study how their interactions affect the system resilience. We show that entangling data in a sliding window limited to the recent past bounds the collateral damage required to irrecoverably destroy a document. We also provide evidence that entanglement chosen uniformly at random forces an attacker who wants to irrecoverably destroy a document archived long enough to destroy a constant fraction of all document archived after it.

4) Practical Considerations: In the data entanglement work of [14] and [15], all-or-nothing integrity only works with static data stores. Furthermore, the proposed solution based on polynomial interpolation is too complex for a practical implementation in its current form. We emphasize that our objective is to achieve both data integrity and censorship/tamper resistance in a way implementable in practical systems. Thus, we use practical constraints that keep an actual implementation realistic while being simple enough to allow analysis. We aim for solutions that archive a new document using only a small constant amount of data already archived and without modifying it, and read archived documents using a small constant amount of archived data. All our underlying assumptions and design choices are implementable using state-of-the-art and optimized coding and storage techniques.

C. Publication Note

We presented a preliminary version of this work at the 2015 International Symposium on Information Theory [31]. This extended version includes additional material such as the hardness of the optimal attack, additional suboptimal attacks, extended simulations, the proofs, implementation details, and a comprehensive discussion of the security and implementation strengths and weaknesses of the architecture.

D. Outline

The rest of this article is organized as follows. Our novel storage architecture is formally described in Section II, followed by assumptions and architecture constraints in Section III. We present our recovery algorithm in Section IV. We analyze the optimal attack in Section V, and describe suboptimal attacks in Section VI. Two entanglement strategies, proximity entanglement, and uniformly random entanglement, are respectively studied in Sections VII and VIII. In Section IX, we discuss extensions and general security considerations. Finally, we conclude the paper in Section X.

II. 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 $e \triangleq p - s$ 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 [32] is used to create $p \geq s$ parity blocks which are then archived on the system. The code rate is $\frac{s+t}{s+t+p}$, but since only the parity blocks are archived, the *storage rate* on the physical medium is $\frac{s}{p}$. 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 $e \triangleq p - s$ block erasures per document on the storage medium can be corrected.[1] Note that increasing t does not

[1]Any two of the three parameters s, e, p are sufficient to calculate the other. The parameter e is included for convenience to distinguish the local erasure capability of our scheme, which is different from that of an MDS code used straightforwardly. It is also possible to use a code which is systematic for the old entangled blocks but not for the source blocks. This allows us to puncture the code without decreasing its error-correcting capability since the source blocks must no longer be erased.

```
Algorithm 1: Entanglement and Archival of Document $d_k$
input : $S \leftarrow s$ source blocks of document $d_k$
1 Select $t$ old blocks already archived based on a
  predefined entanglement strategy
2 Apply the MDS code to the $t$ pointer blocks and $s$
  source blocks to generate $p$ parity blocks
3 Archive the $p$ parity blocks
```

increase the storage overhead nor the local error-correcting capability, but does increase coding and decoding complexity. Algorithm 1 summarizes the entanglement and archival of a document d_k. The challenging part of our approach is to choose the pointers to entangled blocks in a way that provides good anti-tampering and data integrity.

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 1. 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 1a, 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 locally from its codeword, 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 $\{20, 21, 22, 24\}$, as illustrated in Figure 1b. 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.

Definition 2: A set of documents forms an integrity set I *if for all the documents in I, at least $e+1$ blocks do not appear in any document of the complementary set $\overline{I}$. We write $I(d_k)$ to express that a document d_k belongs to I.*

If an attacker wants to irrecoverably censor a document d_k, he must partition the set of documents in two: an integrity set $I(d_k)$ of corrupted documents including d_k, each with at least $e + 1$ erased blocks,[2] and the complementary set $\overline{I(d_k)}$ of uncorrupted documents without any erased block.

Definition 3: Let $\mathcal{A}$ be the set of all (s, t, e, p)-archives with $K \geq k$ archived documents and fixed $s, t, e,$ and p.[3] *We write $I_{min}(d_k)$ to denote the size of the smallest integrity set of document d_k for a fixed archive $a \in \mathcal{A}$, $\mathcal{I}_{min}(d_k) \triangleq \min_{1\leq j\leq k}(I_{min}(d_j))$ for the size of the smallest of the integrity sets of the first k documents for a fixed archive $a \in \mathcal{A}$, and ${}_{max}\mathcal{I}_{min}(d_k) \triangleq \max_{a\in\mathcal{A}}(\mathcal{I}_{min}(d_k))$ for the largest $\mathcal{I}_{min}(d_k)$ over all possible archives $a \in \mathcal{A}$. We write $\mathcal{I}_{min}$ and ${}_{max}\mathcal{I}_{min}$ when $K \gg k$.*

Note that $\mathcal{I}_{\min}(d_k)$, ${}_{\max}\mathcal{I}_{\min}(d_k)$, $\mathcal{I}_{\min}$ and ${}_{\max}\mathcal{I}_{\min}$ are non-decreasing functions of k. The dependency between documents is not symmetric: if the smallest integrity set containing document d_k also contains document d_l, the smallest integrity set containing d_l does not necessarily contain d_k, and $I_{\min}(d_l) \leq I_{\min}(d_k)$.

Our goal is to ensure that the smallest integrity set is as large as possible for every document. If the smallest integrity set is large, we guarantee data integrity, tamper resistance and censor resistance at the same time: every document d_k is as securely and reliably archived as the smallest integrity set containing it. A large $\mathcal{I}_{\min}$ with $K \gg k$ guarantees that all *old enough* documents have good integrity.

Another relevant parameter is the size of the *integrity window* required to irrecoverably delete a document. The integrity window of an integrity set $I = \{d_i, d_{i_1}, d_{i_2}, \ldots, d_j\}$ where $i < i_1 < i_2 < \cdots < j$ is $W \triangleq \{d_i, d_{i+1}, d_{i+2}, \ldots, d_j\}$, and its size is $j-i+1$. It is the number of documents, including documents that are not deleted, from the oldest to the most recent document of a given integrity set.

Definition 4: Let $\mathcal{A}$ be the set of all (s, t, e, p)-archives with $K \geq k$ archived documents and fixed $s, t, e,$ and p.[4] *We write $W_{min}(d_k)$ to denote the size of the smallest integrity window of document d_k for a fixed archive $a \in \mathcal{A}$, $\mathcal{W}_{min}(d_k) \triangleq \min_{1\leq j\leq k}(W_{min}(d_j))$ for the size of the smallest of the integrity windows of the first k documents for a fixed archive $a \in \mathcal{A}$, and ${}_{max}\mathcal{W}_{min}(d_k) \triangleq \max_{a\in\mathcal{A}}(\mathcal{W}_{min}(d_k))$ for the largest $\mathcal{W}_{min}(d_k)$ over all possible archives $a \in \mathcal{A}$. We write $\mathcal{W}_{min}$ and ${}_{max}\mathcal{W}_{min}$ when $K \gg k$.*

From Figure 1b, we can see that the attack in red is optimal, thus $I_{\min}(d_5) = 3$ and $W_{\min}(d_5) = 4$. In fact, it is not hard to show that irrecoverably destroying any of the first 5 documents of the archive requires the deletion of blocks from at least three documents in a window of size at least 4, thus $\mathcal{I}_{\min}(d_5) = 3$ and $\mathcal{W}_{\min}(d_5) = 4$. $\mathcal{I}_{\min}(d_6) = 2$ since d_6 can be destroyed by deleting the blocks $\{16, 17, 18, 26\}$ in documents d_6 and d_9.

III. Model Assumptions and Architecture Constraints

A. Storage Backend Requirements

We intend STEP-archives to be used on top of existing available storage services, and we make a number of assumptions about the behaviour of the storage services used as the backends for data and metadata. We examine the consequences of relaxing these assumptions in Sections III-C, IX and X.

[2]From our definition of integrity set, it is possible to delete $e + 1$ blocks for every document in I without causing damage outside I.

[3]What distinguishes the archives in $\mathcal{A}$ is which tangled blocks are selected for each document.

[4]See Footnote 3.

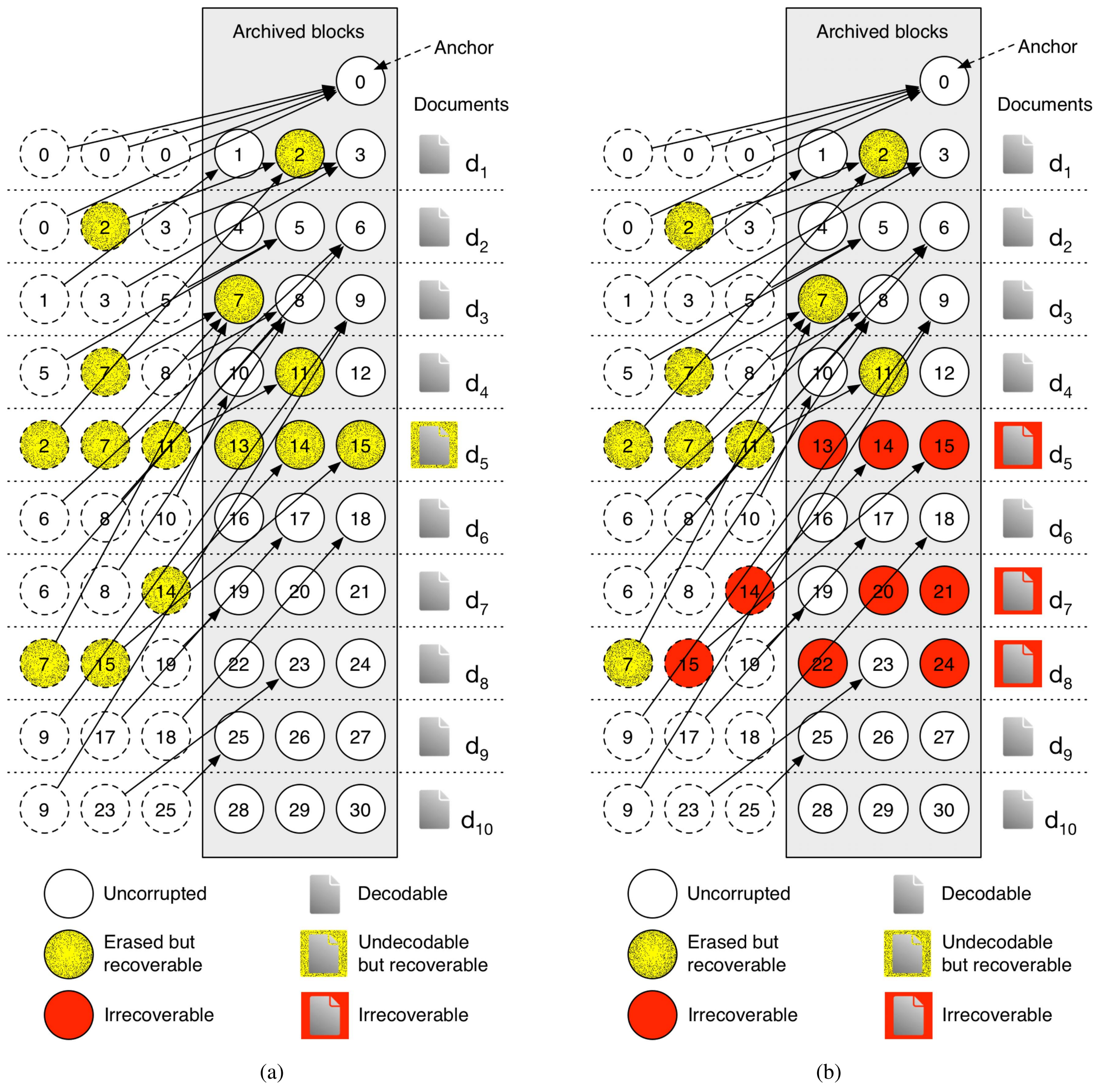


Fig. 1. (1,3,2,3)-archive. 4 out of 6 blocks are required to recover a document. (a) Blocks {2, 7, 11, 13, 14, 15} are erased. (b) Blocks {2, 7, 11, 13, 14, 15, 20, 21, 22, 24} are erased.

- We use an immutable key-value store as the underlying structure for storage of data. In this context, updates to old documents effectively become new documents. This is a reasonable assumption for a large class of applications, and has great practical benefits. In addition, we require that the storage interface to the data provides self-integrity; that is, when fetching a stored block B using the corresponding storage key, the system must either return the same block B or an error; it cannot return another data block $B' \neq B$. In practice, this is achieved by having the storage key computed as a cryptographically secure (collision-resistant) hash function of B, and have the storage client validate the hash from the data against the storage key for every read operation. A malicious server must be able to break the hash function in order to break this self-integrity.
- We assume that the data store uses blocks of a fixed size. Padding and splitting large documents into smaller pieces is left to the layers above our construction.
- We allow malicious servers to refuse to complete arbitrary data requests, however, we require stronger integrity for metadata. We assume that the adversary is not able to tamper with existing metadata at all. Thus, after an attack (or accidental failure), the system knows with certainty which data blocks have been corrupted or tampered with, i.e., the only errors at this level are erasures.
- There are no confidentiality constraints for the data, nor for the metadata. On the other hand, we make no confidentiality guarantees either; proper usage of encryption is again left to the upper layer. We assume that the adversary knows which data blocks belong to which document. The attacker can also observe the creation date of each data block and can therefore order them chronologically if required.
- In practice, popular cloud storage solutions, like Amazon S3 or Microsoft Azure Blob Storage, can be used for the data storage. A decentralized cloud storage system similar

to the Sia [33] and Storj [34] projects could also be of interest.

B. Forward Integrity

With immutable data structures such as STEP-archives, all-or-nothing integrity as defined in [14] and [15] is unattainable because the best system will at most guarantee that data can only be tampered with by destroying all data stored *after* it. Although this can be seen as an undesirable property, we argue that we cannot achieve all-or-nothing integrity without violating our practical requirements. The best we can hope for in this setting is **forward integrity**. Digital storage capacity and usage have increased at an exponential rate for the last 40 years and might do so for the foreseeable future. This allows the possibility to provide comprehensive forward integrity quickly on a time scale.

We note that all-or-nothing forward integrity is possible with STEP-archives using a non-constant number of pointers per document: on an archive with $s = 1$ and $e \geq 1$, 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. In other words, for every k, document d_k is recoverable if and only if all the documents archived after d_k are recoverable. 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. Although one might imagine interesting settings in which t grows logarithmically with the size of the archive, in this work we target highly efficient solutions. We thus focus on archives with t constant and small, and attempt to quantify the amount of forward integrity they can provide.

We again emphasize that with STEP-archives the integrity, tamper-resistance and censorship-resistance of any document are equivalent and correspond to the size of the largest integrity set containing it. To destroy a document, we must recursively destroy other documents. This is the dual of the architecture of [14] and [15], for which destroying a small amount of data recursively destroys all the data.

C. Trust Model

Our goal is to offer a storage system providing forward integrity, allowing tradeoffs between security and performance with different deployment models. In a trusted private cloud where the clients trust the storage provider to behave correctly, the entanglement, encoding and decoding, scrubbing,[5] and metadata bookkeeping, can be performed by the storage system itself. In this setup, the STEP-archive is a simple redundant storage system, with attractive robustness and offline censorship and tampering resistance. Malicious clients cannot force an honest storage provider to misbehave and thus cannot alter the entanglement properties of the archive. The system therefore resists censorship by destruction of stored data, but not censorship by malicious alteration of its behaviour.

[5]Background recovery of corrupted data.

Without a trusted infrastructure, we can guard against insider censorship and tampering resistance by assuming that the adversary has full control over the behaviour of the data storage system. The entanglement, encoding, decoding, scrubbing, and metadata bookkeeping must thus be performed by the clients, who might not trust each other. The clients check the integrity of all blocks and codewords read from the storage system before performing operations on them. Compared to the previous scenario, writes have a computational and bandwidth overhead (the clients need to fetch at least the t pointers before encoding a document locally). In an untrusted environment, one possibility is to store the metadata in a blockchain. Our metadata is immutable, and therefore perfect for such storage. Cryptographically signing it would allow clients to build a reputation, as anyone can verify the soundness of metadata, at the cost of fetching and decoding the relevant blocks.

Even within a blockchain, malicious clients could perform two classes of online attacks. The first is to store large amounts of irrelevant data, lowering the effective impact of the collateral damage mandated by censorship resistance. This cannot be avoided, but at least comes with a significant storage cost for the adversary. The second is to store unsound metadata, i.e., either invalid metadata or valid metadata corresponding to invalid codewords that do not actually allow recovery of concerned erased blocks. In our mathematical analysis, we assume that an attacker wishes to perform an offline attack on a well-constructed STEP-archive without bogus documents. We discuss the two online attacks and their countermeasures in Section X-D.

IV. Reconstruction Algorithm

One of the interesting aspects of our approach is its asymmetry: while it is impossible for an attacker to find the optimal strategy to irrecoverably tamper with a target document d_k in polynomial time (unless $\mathbf{P} = \mathbf{NP}$), repairing the system if the damage done is recoverable is easy and doable in linear time, as shown in Algorithm 2. The idea is first to scan the archive and build the set C of corrupted documents with at most e erased blocks. We can then take any document d in C, remove it from C, decode it, recover its erased blocks, and update C by adding the corrupted documents, if any, that previously had strictly more than e erased blocks but that now have at most e. The algorithm stops when C is empty, at which point either all the erased blocks are recovered or all the remaining corrupted documents have more than e erased blocks. The following lemma is straightforward.

Lemma 5: Let B be the set of erased blocks, and C the corresponding set of corrupted documents. The set of documents R irrecoverable by the reconstruction algorithm is the largest integrity set $I \subseteq C$ whose set of erased blocks is a subset of B.

Proof: By design of the reconstruction algorithm, R is an integrity set whose set of erased blocks is a subset of B. It is clear that $R \supseteq I$ because the reconstruction will never be

Algorithm 2: Reconstruction Algorithm

```
input : $e \leftarrow$ erasure decoding capability of the code
        $C \leftarrow$ set of corrupted documents in the archive
1 while $C \neq \emptyset$ do
2     pick an element $d \in C$
3     decode $d$ and recover its set of erased blocks $B$
4     forall the archived documents $d'$ with at least one block in $B$ do
5         if $d'$ is corrupted and has at most $e$ erased blocks then
6             $C \leftarrow C \cup \{d'\}$
7     $C \leftarrow C \setminus \{d\}$
```

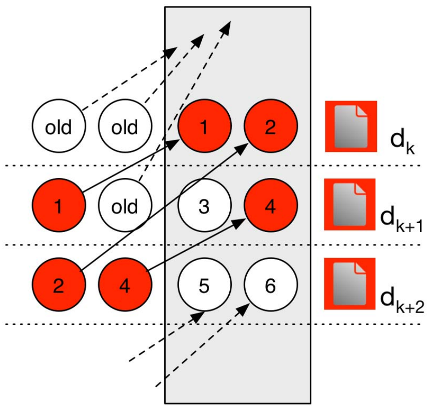


Fig. 2. (1,2,1,2)-archive with dependency cycle.

able to recover any document in I. It is also clear that $R \subseteq I$, otherwise $R \cup I$, which is an integrity set whose set of erased blocks is a subset of B, would be larger than I. □

In Figure 1a, the reconstruction algorithm can recover all the erased blocks, whereas in Figure 1b it can recover the yellow blocks $\{2, 7, 11\}$ but is incapable of recovering the red blocks $\{13, 14, 15, 20, 21, 22, 24\}$. The algorithm is highly parallelizable: in Figure 1a, it can recover blocks $\{2, 7, 11, 14, 15\}$ in parallel, after which it recovers Block 13 in a second step.

In certain cases, it is possible to recover the documents in an integrity set with more than e erased blocks per document. Consider the small section of a $(1, 2, 1, 2)$-archive shown in Figure 2. If an attacker erases blocks $\{1, 2, 4\}$, none of the three documents can be decoded and recovered by itself. However, we can easily find a code (linear or nonlinear) such that there is only one solution for the three erased blocks that results in three valid codewords. This occurred because blocks $\{1, 2, 4\}$ form a dependency cycle between the three documents and the attacker only erased blocks belonging to that cycle. However, even if there is only one solution for each erased block, the reconstruction algorithm will fail and reconstruction codeword per codeword in not possible.[6] Furthermore, a single erased block that is not constrained by other documents is sufficient to ensure multiple solutions. In Figure 2, the attacker could also have erased Block 3, which breaks the dependency cycle by adding a degree of freedom.

[6]In principle a grouped recovery could be attempted, at a greatly increased cost as the algorithm now needs to find all cycles in the block-document graph. Furthermore, the attack algorithms we present later can easily be tweaked to avoid cycle formation, at the expense of solution quality.

V. Optimal Attack

The most natural way to represent a (s, t, e, p)-archive is as a $(t+p)$-uniform hypergraph $H^* = (V^*, E^*)$, where the set of vertices V^* is the set of all archived blocks, and each document d_k corresponds to a hyperedge in E^*. However, the dual $(t + p)$-regular hypergraph $H = (V, E)$ is more conducive to analysis, thus it is the model used in this section. In this setting, the set of vertices V is the set of all archived documents, and each archived block corresponds to a hyperedge in E. Finding the best attack to censor a document d_k irrecoverably corresponds to finding the minimum subhypergraph V' of minimum degree at least $e + 1$ containing d_k. This subhypergraph V' corresponds to the smallest integrity set $\mathcal{I}_{\min}(d_k)$.

For simple undirected graphs, the problem of finding the *minimum subgraph of minimum degree* $\geq d$ *(MSMD$_d$)*, also called d-*girth*, has a long history starting from the work of Erd́os *et al.* [35] and Bollobás and Brightwell [36]. More recently, [37] proved that for $d \geq 3$, MSMD$_d$ is **NP**-hard and cannot be approximated within a constant factor in polynomial time if $\mathbf{P} \neq \mathbf{NP}$. This was improved in [38], where the authors showed that for $d \geq 3$ and $\epsilon > 0$ there is no polynomial-time algorithm with approximation ratio $2^{\mathcal{O}(\log^{1-\epsilon} n)}$ unless $\mathbf{NP} \subseteq \mathbf{DTIME}\left(2^{\mathcal{O}(\log^{1-\epsilon} n)}\right)$, even with graphs with degree d or $d + 1$. The authors also presented a polynomial-time randomized approximation algorithm with ratio $\mathcal{O}\left(\frac{n}{\log n}\right)$, and a brute-force polynomial-time deterministic approximation algorithm of ratio $\mathcal{O}\left(\frac{n \log n}{\log \log n}\right)$ for low-degree graphs. The parametrized complexity of the d-girth problem was studied in [39], where the authors proved that the problem is **W[1]**-hard[7] for general graphs, but fixed-parameter tractable when graphs have bounded local tree width. From the discussion in [38], optimization of the d-girth problem for $d \geq 3$ appears very hard, in the vein of other very hard problems to approximate like *maximum clique*, *chromatic number* and *longest path*, and the authors conjecture that

> there is no polynomial-time approximation algorithm for the d-girth problem with ratio $n^{1-\delta}$ for some constant $\delta > 0$ unless $\mathbf{P} = \mathbf{NP}$.

We note that for $d = 2$, the problem is the standard girth of a graph and corresponds to the length of its shortest cycle, which is solvable efficiently by dynamic programming.

In this section, we consider a target document d_k archived long enough, i.e., with $K \gg k$ documents archived after it. We mention that for general hypergraphs, contrary to graphs, the 2-girth problem seems difficult as well, however we show that under certain conditions, there exists a polynomial-time algorithm for the optimal attack on (s, t, e, p)-archives with $e = 1$. To prove this result, we present a polynomial-time algorithm to calculate the 2-girth of multigraphs with loops, which appears to be new. We then show, using a reduction to

[7]Consult [40] for a formal definition.

the $e+1$-girth problem, that for $e \geq 2$ and $t \geq e+2$, finding the optimal attack targeting document d_k is **NP**-hard, impossible to approximate within a constant factor in polynomial time if $\mathbf{P} \neq \mathbf{NP}$, and impossible to approximate with ratio $2^{\mathcal{O}(\log^{1-\epsilon} n)}$ unless $\mathbf{NP} \subseteq \mathbf{DTIME}\left(2^{\mathcal{O}(\log^{1-\epsilon} n)}\right)$.

Lemma 6: There exists a polynomial-time algorithm to calculate the 2-girth of a multigraph with loops.

Proof: Consult Appendix A. □

Corollary 7: If each block of a $(s, t, 1, p)$-archive belongs to at most two documents, then there is a polynomial-time algorithm to find the smallest integrity set containing document d_k.

Proof: If each block belongs to at most two documents, then each hyperedge has one or two vertices, thus the hypergraph is a multigraph with loops. Since $e = 1$, the smallest integrity set of this multigraph is its 2-girth which from Lemma 6 can be found in polynomial time. □

Theorem 8: Let $\epsilon > 0$, and consider a (s, t, e, p)-archive with $e \geq 2$, $t \geq e+2$, and $K \gg k$ documents archived after a target document d_k. Finding $I_{min}(d_k)$ is **NP***-hard. Furthermore, approximating $I_{min}(d_k)$ within constant factor in polynomial time is impossible if $\mathbf{P} \neq \mathbf{NP}$, and approximating $I_{min}(d_k)$ with approximation ratio $2^{\mathcal{O}(\log^{1-\epsilon} k)}$ in polynomial time is impossible unless $\mathbf{NP} \subseteq \mathbf{DTIME}\left(2^{\mathcal{O}(\log^{1-\epsilon} k)}\right)$.*

Proof: Consult Appendix A. □

Note that it follows from the reconstruction algorithm that verifying whether a set of documents is an integrity set is in **P**, hence finding $I_{\min}(d_k)$ is **NP**-complete.

VI. Suboptimal Attacks

Because we do not know any good polynomial-time algorithms to optimally solve the girth problem relevant to attacking our system (or even to find a good approximate solution), we turn to more specific techniques, taking the special structure of our archive into account. In this section we consider several linear-time heuristics, and use them in later sections to study entanglement strategies.

All the heuristics formulate the attack as a search problem on a tree of partial solutions. A partial solution consists of a set of target documents we are currently committed to destroy, and a set of erased blocks. A solution is complete if the set of erased blocks is sufficient to make the target document set irrecoverable, more precisely an integrity set with at least $e + 1$ erased blocks per document. A partial solution must be completed by deleting some blocks referenced by recoverable documents. To make sure the target document set is not recoverable, no destroyed blocks must be referenced by documents outside of the target set; every time we choose to destroy a new block, we must commit to destroy all documents referencing it.

From a partial solution, every possible choice of new blocks to erase gives a new partial (or possibly complete) solution, forming a tree of solutions. For the initial solution, we take the set of documents to censor, along with a (yet) empty set of erased blocks.

Observe that, except at the start of the algorithm, the target set can be unambiguously computed from the set of erased blocks. The tree is, in fact, a lattice, as two different paths may lead to the same partial solution, differing only in order of processing. This lattice is finite, having at its other extremity the worst case where the entire system has been erased.

Algorithm 3: Greedy Attack Framework

```
input : $k \leftarrow$ the index of the target document
        $e \leftarrow$ erasure decoding capability of the code
output: $B$// set of erased blocks making $d_k$ irrecoverable
note  : documents($b$) is the set of documents with $b$ as a block
1  $R \leftarrow \{k\}$        // Corrupted but decodable targets
2  $C \leftarrow \{k\}$        // All corrupted targets
3  $B \leftarrow \emptyset$        // Erased blocks
4  while $R \neq \emptyset$ do
5      $r \leftarrow \min(R)$
6      while $d_r$ has fewer than $e+1$ erased blocks do
7          $score \leftarrow \emptyset$
8          forall the blocks $b$ not erased in $d_r$ do
9              score[$b$] $\leftarrow$ heuristic($C, B, b$)
               // Available heuristics:
               // MinimumAttack, LeapingAttack,
               // CreepingAttack, TailoredAttack
10         $b \leftarrow \arg\min_b$ score[$b$]
11         $B \leftarrow B \cup \{b\}$
12         $R \leftarrow R \cup$ (documents($b$) $\setminus C$)
13         $C \leftarrow C \cup$ documents($b$)
14     $R \leftarrow R \setminus \{r\}$
15 return $B$
```

A. Greedy Attacks

The simplest way to explore this lattice is with a greedy algorithm. We iteratively walk down the lattice of solutions along a single path, eventually reaching a complete solution. The greedy attack framework is described in Algorithm 3. For convenience, we maintain the set C of all corrupted targets and the set R of corrupted but decodable targets. The attack is complete when R becomes empty, at which point all the documents in C are irrecoverable. Until then, we iterate over R in chronological order, and for each document in it, we erase the minimal[8] number of blocks required to make the processed document impossible to decode locally. As there will most of the time be more candidate blocks for deletion than the minimum required, we use one of four simple heuristics to choose the blocks to delete. This leads to four variants of a greedy attack: minimum attack, leaping attack, creeping attack, and tailored attack.

1) Minimum Attack: The minimum attack, described in Algorithm 4, minimizes the set C of corrupted target documents, by always preferring blocks that are referenced

[8]This is dictated by the code parameters and the number of blocks already erased in earlier steps.

Algorithm 4: MinimumAttack(C, B, b)

```
1 return $|C \cup documents(b)|$
```

Algorithm 5: LeapingAttack(C, B, b)

```
1 $N \leftarrow documents(b) \setminus C$
2 if $N = \emptyset$ then
3     return $-\infty$ ;                    // free block
4 else
5     return $-\min(N)$
```

by the least amount of documents not already in the target set.

2) Leaping Attack: The leaping attack, described in Algorithm 5, is based on the intuition that it is easier to attack recent documents than older ones. We show in Section VIII that the leaping attack is especially suitable to damage a system built with entangled blocks chosen uniformly at random. The score of each block is the negation of the timestamp of the oldest document in $\overline{C}$ to reference it, with higher timestamps more desirable. Intuitively, we try to leap over documents by moving $\min(R)$ forward in time as fast as possible toward the end of the archive. When all documents referencing a block are already in C, the block score is the minimum of an empty set of integers, which we take as negative infinity. In this case, the block is *free* and can thus be erased without propagation to uncorrupted documents. Thus, free blocks are always erased in priority (free blocks are implicitly deleted in priority for the minimum attack as well).

Observe that a document can always be made undecodable by deleting all its parity blocks, without having to delete any pointers. This is because, as defined in Section II, the number of parity blocks p is strictly greater than the number of correctable errors e.

Also, the oldest document referencing a block is always the primary document of this block. It follows that with the leaping heuristic, parity blocks are always favored over pointers unless all the older documents using a pointed block are already in C.

3) Creeping Attack: The creeping attack, described in Algorithm 6, intuitively tries to keep the set of corrupted documents C as compact as possible in time. We will see in Section VII that the creeping attack is effective against window-based entanglement strategies. The creeping attack is similar in essence to the dual of the leaping attack. However, simply mirroring the min-max behavior of the leaping attack does not give a useful algorithm, as it can cause C to grow fast and far in the past. We could explicitly forbid the algorithm from targeting documents older than the initial target (this constraint is implicit in the leaping attack), but instead we address this shortcoming by minimizing the range of C. Although this is not explicitly written in the algorithm, when two blocks have the same cost, the algorithm erases the block with the smallest $|C'|$. Thus, as for the leaping attack, the free blocks that do not propagate to other documents are erased in priority.

4) Tailored Attack: Although we study pointers selected uniformly at random in Section VIII, here we briefly

Algorithm 6: CreepingAttack(C, B, b)

```
1 $C' \leftarrow C \cup \text{documents}(b)$
2 return $\max(C') - \min(C')$
```

Algorithm 7: TailoredAttack(C, B, b)

```
  input: $t$ ;          // Number of pointers per
         document
  input: $K$ ; // Number of archived documents
  note : documents($b$) is the set of documents with $b$ as a
         block
         blocks($c$) is the set of blocks of document $d_c$
1 cost $\leftarrow 0$
2 $B' \leftarrow B \cup b$
3 $C' \leftarrow C \cup \text{documents}(b)$
4 forall the $c \in C'$ do
5     yettoerase $\leftarrow \max(0, e + 1 - |B' \cap \text{blocks}(c)|)$
6     cost $\leftarrow$ cost $+ 1 +$ yettoerase $\cdot \ln\left(1 + \frac{K-c}{c}\right)^t$
7 return cost
```

mention that we can calculate the expected number of times a parity block is used as a pointer by younger documents (Lemma 14). This allows us to estimate, when we erase a block, the propagation of the attack to all the documents used by that block. This attack, tailored to uniform random entanglement, is described in Algorithm 7 and further discussed in Section VIII.

B. Depth-First Search

The greedy algorithms are fast since their complexity is linear in the number of archived documents. We implemented a recursive depth-first search over the tree of partial solutions using our four heuristics. The first complete solution produced always matches the output of the greedy attack. The tree is then backtracked, looking for smaller integrity sets. Since the cost function $|C|$ is nondecreasing as we go down the tree, we can perform branch pruning as soon as the partial cost exceeds the cost of the currently best known solution.

This algorithm thus offers a tradeoff between time and solution quality, at the expense of increased memory usage. Figure 3 shows the progression of solution quality over time for twelve randomly selected archives. Unfortunately, as shown in this sample run, even with pruning depth-first search is expensive and does not always progress to the optimal solution quickly. Solution quality[9] is not proportional to the time spent attacking. Instead, the solution improves in unpredictable large steps. Intuitively, this can be explained by the fact that more recent documents are less protected, and much easier to attack. By searching depth-first, we spend a lot of effort trying to optimize the later stages of the attack, which may already be close to optimal, whereas the decisions with the most impact on the overall cost are the ones taken at the beginning of the attack.

[9]as the inverse of the size of the integrity set.

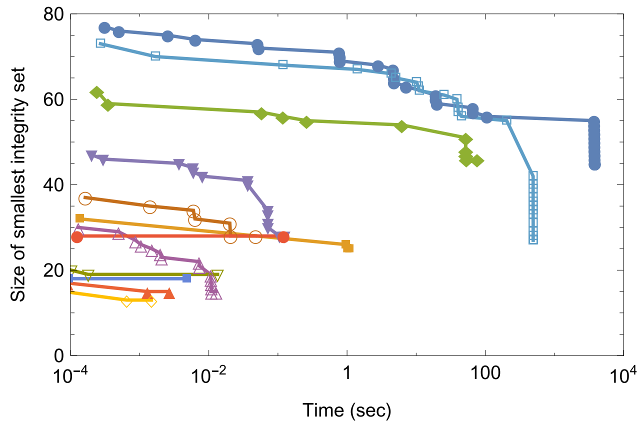


Fig. 3. Trace of the depth-first search leaping attacks for twelve $(1, 3, 2, 3)-$archives with 10^4 documents, target document d_{1000} and pointers chosen uniformly at random. The algorithm traversed the entire tree in $\approx 10^4$ seconds in the worst case.

C. Bounded Breadth-First Search

The inefficiency of depth-first search motivates the investigation of bounded breadth-first search algorithms. For large systems, it is practically infeasible to traverse the entire solution tree, and bounded breadth-first search algorithms converge much faster than depth-first search protocols. We thus keep a collection of partial attack states, ranked according to some of our heuristics, and expand the most promising partial solutions first. We expand all the partial solutions into their child states at once, then only retain the best ones, up to the selected buffer size. We thus deal with a series of sets of solutions, for which all solutions in the same set are located at the same depth in the tree. This simplifies the analysis of the behavior: we can enforce a constant maximum width, for all depths, on the subtree we are exploring. We cannot apply the same pruning strategy as in the depth-first search, because no complete solution is known before the end of a run, but we can control how much time we spend in the most critical part of the search tree.

VII. Proximity Entanglement

When a document is not being pointed to, it can always be tampered with, without propagation. Thus, it makes sense to ensure that a new document will be quickly pointed to. A potential solution is to choose the entangled blocks using a sliding window bounding the pointers to documents in the recent past. We show in this section that this approach has the drawback that an attacker can irrecoverably tamper with documents, with an efficient attack to do so, concentrated in the close vicinity of the target document.

Definition 9: We define the entangled and parity blocks of document d_k for a (s, t, e, p)-archive as

$$d_k \triangleq (t_k^1, t_k^2, \ldots, t_k^t, b_k^1, b_k^2, \ldots, b_k^p).$$

Consider a (s, t, e, p)-archive with a sliding window of size w, i.e., the pointers in document d_k do not point to documents older than d_{k-w}. The first thing to consider is the number of pointers per document. If $t < p$, in other words if the number of pointers per document is smaller than the number of archived blocks per document, then with a sliding window some blocks will never be pointed to. These unprotected blocks can thus be deleted without propagation to blocks from other documents. If $t = 1$, for instance, then ${}_{\max}\mathcal{I}_{\min} = 2$ and ${}_{\max}\mathcal{W}_{\min} = w + 1$. This can be achieved by taking the single entangled block of document d_k from the parity blocks of document d_{k-w}. Using this structure, an attacker who wants to delete d_k irrecoverably can do so by deleting the parity blocks of d_k and the unprotected parity blocks of d_{k+w}. It is possible to obtain a larger minimum integrity set and/or a larger minimum integrity window for some documents, but in this case other documents will remain completely unprotected.

We now increase the number of pointers per document to $t = p$ and organize the pointers so that every block is pointed to at most once. We show with the next lemma that the size of the best integrity windows increases but remains bounded.

Lemma 10: Consider a (s, t, e, p)-archive with $p \geq 3$, $t = p$, a sliding window of size $w > p$, and such that every block in the archive is pointed to at most once. Then,

$$2w - p + 1 \leq {}_{max}\mathcal{W}_{min} \leq 2w.$$

Proof: Consult Appendix B. □

Having more than p entangled blocks per document with a fixed sliding window does not appear to provide much more integrity, and even becomes harmful as t increases further, as shown next.

Lemma 11: Consider a (s, t, e, p)-archive with a sliding window of size w and $t = p{\cdot}w$ entangled blocks per document. Then,

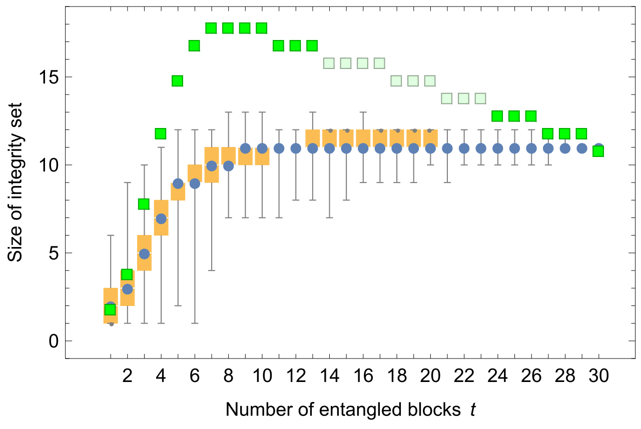
$${}_{max}\mathcal{I}_{min} = {}_{max}\mathcal{W}_{min} = w + 1.$$

Proof: Since $t = p \cdot w$, the entangled blocks of every document must point to every parity block of every document in its sliding window. For the first w archived documents, the entangled blocks that should be pointing to documents that do not exist do instead point to an anchor block. Since the entanglement structure is the same for every document d_k, we can without loss of generality attack d_k by tampering with documents archived after d_k. Thus, to erase d_k, at least $e+1$ of its p parity blocks must be erased. Since we use every block of d_k as an entanglement block in documents $d_{k+1}, \ldots, d_{k+w}$, this is sufficient to erase d_k irrecoverably. □

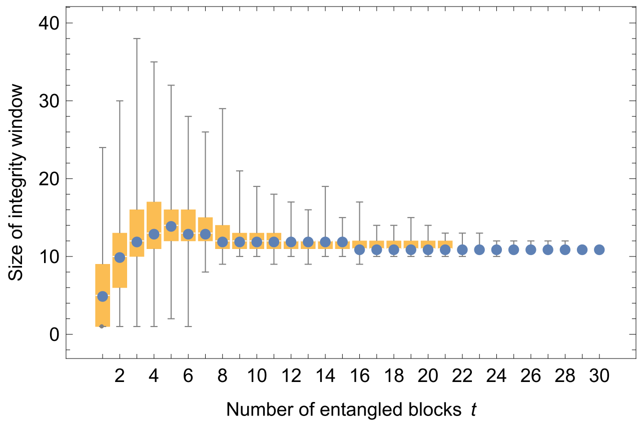
If w is small, a malicious customer is capable of censoring a newly archived document d_k at anytime in the future by archiving $2w - 1$ junk documents immediately after d_k. Allowing several pointers to point to the same parity block does not appear to make this attack more difficult: if the pointers are bounded to w documents in the past, then the anti-tampering protection is also bounded.

A. Random Entanglement Within a Sliding Window

With a small sliding window, we can explore the search tree exhaustively. To illustrate the previous results, Figure 4 shows the result of the optimal attack over $(1, t, 2, 3)$-archives with 10^5 documents, $t \in \{1, 2, \ldots, 30\}$ and entangled blocks chosen at random in a sliding window of size $w = 10$. The attack targets document d_{1000}. For each value of t, the figures show



(a)



(b)

Fig. 4. Optimal attack for 1000 $(1, t, 2, 3)$-archives with a sliding window of size $w = 10$ and $t \in \{1, 2, \dots, 30\}$. The archive contains 10^5 documents, and the attack targets d_{1000}. The box-and-whisker plots show the minimum, first quartile, median (in blue), third quartile, interquartile range (in orange) and maximum across all simulation runs. (a) Size of the smallest integrity set $I_{\min}(d_{1000})$. The green squares represent the size of the smallest integrity sets for the best regular entanglement within the window (see Section VII-B). (b) Size of the integrity windows for the integrity sets found in Figure 4a.

a box-and-whisker[10] plot for 1000 simulations. Figure 4a shows how the integrity sets vary, whereas Figure 4b focuses on the integrity windows. The green squares in Figure 4a represent the best regular entanglement within the window (see Section VII-B). The figures illustrate Lemmas 10 and 11: the median integrity size (integrity window) first increases quickly and then decreases to $W_{\min}(d_{1000}) = I_{\min}(d_{1000}) = w + 1 = 11$ as the number of pointers increases. The large maximal integrity windows for small t do not contradict the lemmas, because our optimal algorithm minimizes the size of the integrity set without considering the integrity window.

Figure 5 shows the result of $\min\limits_{990 \le k \le 1010} I_{\min}(d_k)$ with the same simulation parameters. The attack sequentially targets d_i for $990 \le i \le 1010$, and shows the size of the smallest integrity set among the targeted documents. Compared to Figure 4, we can see that with few pointers, the probability that one of the documents in the interval is weakly protected is high. As the number of pointers increases, the protection from document to document becomes more uniform, and Figures 4 and 5 have the same behavior.

Figures 6 and 7 respectively show the result of the creeping and leaping attacks for the same system parameters, respectively. It shows the general efficiency of the creeping attack, and the inefficiency of the leaping attack when t is small. With large t, however, the randomness in pointer selection disappears and both greedy attacks behave like the optimal attack.

B. Regular Entanglement Within a Sliding Window

Instead of choosing the pointers randomly within the sliding window, we can use the same entanglement structure for each document. We call this strategy *regular entanglement*. For instance, with $t = 3$ pointers and a sliding window of size $w = 10$, the regular structure

$$(t_k^1 \triangleq b_{k-1}^1, t_k^2 \triangleq b_{k-3}^2, t_k^3 \triangleq b_{k-10}^3)$$

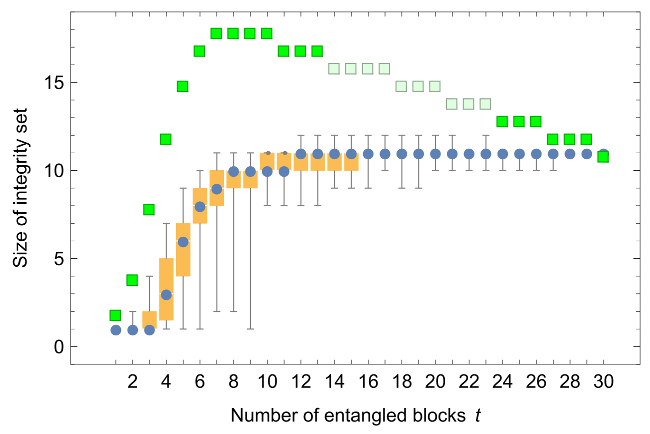
gives ${}_{\max}\mathcal{I}_{\min} = 8$, i.e., the smallest integrity set of any document archived long enough has size eight. This is the best regular structure for $(1, 3, 2, 3)$-archives and $w = 10$, although it is not unique. The green squares in Figures 4a and 5a show the size of the smallest integrity sets for the best regular entanglement structures for $(1, 3, 2, 3)$-archives as the number of pointers per document increases from 1 to 30. The light green squares for $t \in \{14, 15, \dots, 23\}$ are lower bounds since we did not explore all the possible regular structures for these values of t. In this example, regular entanglement is more robust than random entanglement with a small number of pointers, but with a lot of pointers both strategies are equivalent. We mention to conclude this section that the creeping attack always works well for regular entanglement within a sliding window. The leaping attack, however, performs poorly and will generally propagate from the target document to the end of the archive.

VIII. Random Entanglement

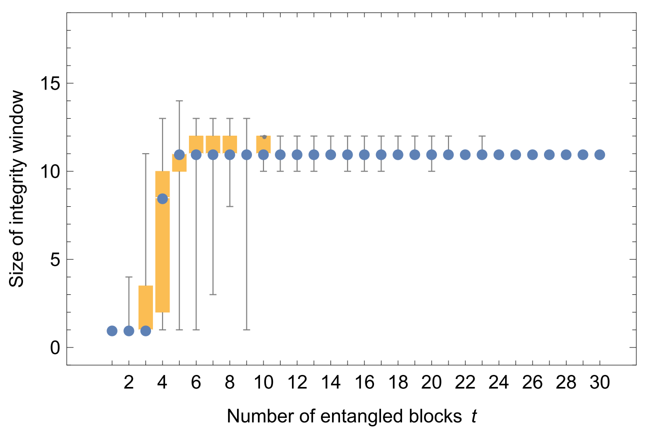
In this section, we study the impact of uniformly random entanglement. In practice, choosing entangled blocks uniformly at random offers two important advantages over highly structured entanglement. First, a structure with randomness prevents the attacker from planning the attack in advance, for instance 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. Conversely, uniformly random entanglement has two drawbacks. The first is that it takes an increasingly longer time to protect young documents as the archive increases. The second drawback, as illustrated in the previous section, is that structured entanglement within a sliding window is much more robust than random entanglement.

Uniformly random entanglement is well-suited for mathematical analysis. We first show a phase transition for the

[10]Showing the minimum, first quartile, median (in blue), third quartile, interquartile range (in orange) and maximum across all simulation runs.

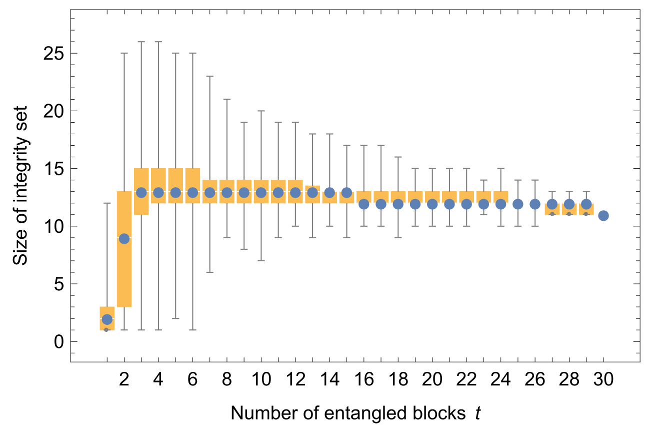


(a)

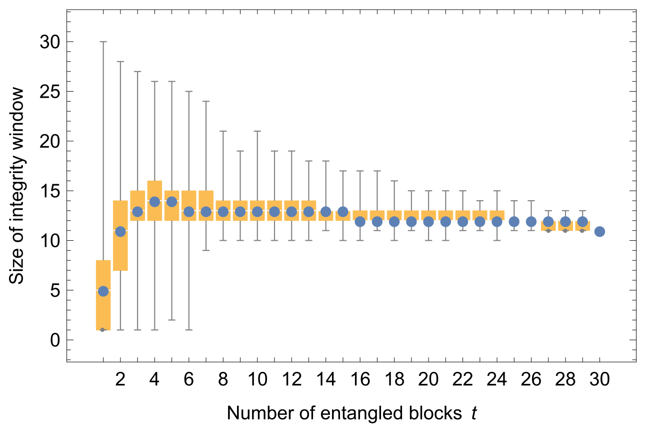


(b)

Fig. 5. Optimal attack for 1000 $(1, t, 2, 3)$-archives with a sliding window of size $w = 10$ and $t \in \{1, 2, \ldots, 30\}$. The archive contains 10^5 documents. (a) Distribution of $\min_{990 \leq k \leq 1010} I_{\min}(d_k)$. The green squares represent the size of the smallest integrity sets for the best regular entanglement within the window (see Section VII-B). (b) Size distribution of the sliding windows for the integrity sets found in Figure 5a.

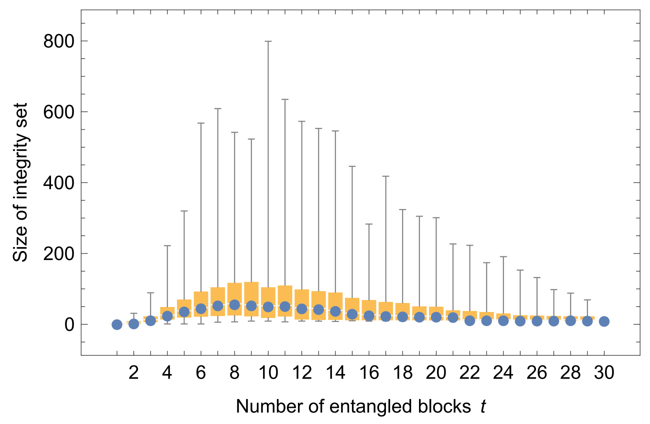


(a)

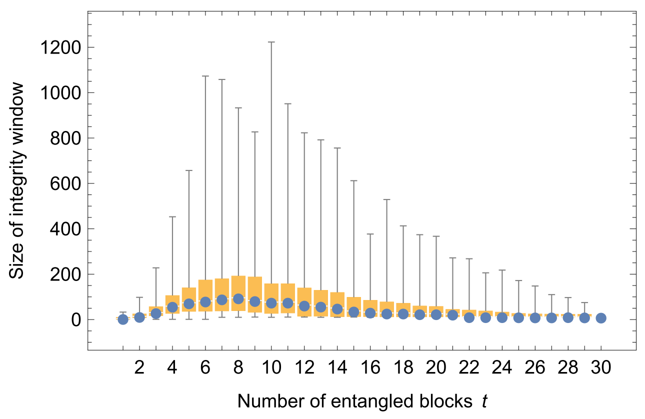


(b)

Fig. 6. Creeping attack for 1000 $(1, t, 2, 3)$-archives with a sliding window of size $w = 10$ and $t \in \{1, 2, \ldots, 30\}$. The archive contains 10^5 documents, and the attack targets d_{1000}. (a) Size of the smallest integrity set $I_{\min}(d_{1000})$. (b) Size of the integrity windows $W(d_{1000})$ for the integrity sets found in Figure 6a.



(a)



(b)

Fig. 7. Leaping attack for 1000 $(1, t, 2, 3)$-archives with a sliding window of size $w = 10$ and $t \in \{1, 2, \ldots, 30\}$. The archive contains 10^5 documents, and the attack targets d_{1000}. (a) Size of the smallest integrity set $I_{\min}(d_{1000})$. (b) Size of the integrity windows $W(d_{1000})$ for the integrity sets found in Figure 7a.

leaping attack as the number of pointers reaches a threshold. Passed that threshold, an attacker who wants to erase a document must corrupt a constant fraction of all documents archived after it. We then provide numerical evidence and conjecture that this phase transition exists for the optimal attack.

Suppose that we want to censor document d_k on a (s, t, e, p)-archive by deleting parity blocks from it. Let $L_i > k$ be defined such that d_{L_i} is the i-th document having a pointer to any of the parity blocks of d_k for $i \geq 1$. If the pointers to entangled blocks are assigned randomly among all the blocks already archived, then the following result can be established.

Lemma 12: If the pointers for a (s, t, e, p)-archive are chosen uniformly at random, then

$$\begin{aligned} E[L_i] &= \infty && \text{if } t = 1; \\ E[L_i] &\sim k\left(1 + \tfrac{1}{t-1}\right)^i && \text{if } t > 1 \text{ and } i \in o(k). \end{aligned}$$

Proof: Consult Appendix C. □

Suppose now that we want to erase a chosen parity block b in document d_k, and let us define $M_i > k$ such that d_{M_i} is the i-th document having a pointer to block b for $i \geq 1$. With pointers chosen uniformly at random, we obtain the following result.

Lemma 13: If the pointers for a (s, t, e, p)-archive are chosen uniformly at random, then

$$\begin{aligned} E[M_i] &= \infty && \text{if } t \leq p; \\ E[M_i] &\sim k\left(1 + \frac{p}{t-p}\right)^i && \text{if } t > p \text{ and } i \in o(k). \end{aligned}$$

Proof: The lemma can be proved directly in a similar fashion as Lemma 12, although this results in a rather cumbersome proof.

Instead, we prove the lemma when t is a multiple of p. For large enough k, the pointer behavior of an archive with p parity blocks and t pointers per document, if t is a multiple of p, is similar to the pointer behavior of an archive with 1 parity block and $t' = \frac{t}{p}$ pointers per document. Hence, from Lemma 12 we obtain

$$\begin{aligned} E[M_1] &\sim k\left(1 + \frac{1}{t'-1}\right)^i \\ &\sim k\left(1 + \frac{1}{\frac{t}{p}-1}\right)^i \\ &\sim k\left(1 + \frac{p}{t-p}\right)^i. \end{aligned}$$

□

Let N_l, $l > 0$, be the number of documents having a pointer to a parity block of document d_k once document d_{k+l} is reached. The following result can be established.

Lemma 14: Consider a (s, t, e, p)-archive with the pointers chosen uniformly at random. If $l \in O(k)$, then

$$E[N_l] \sim \ln\left(1 + \frac{l}{k}\right)^t.$$

Proof: Consult Appendix C. □

Corollary 15: Let i be a positive integer. The expected number of documents in the archive required before the parity blocks of document d_k are pointed to i times is

$$D_i \sim k \cdot \left(e^{\frac{1}{t}}\right)^i.$$

Proof: Using $N_l = i$ in Lemma 14 leads to

$$l \sim k(e^{\frac{i}{t}} - 1),$$

thus

$$\begin{aligned} D_i &= k + l \\ &\sim k + k(e^{\frac{i}{t}} - 1) \\ &= k\left(e^{\frac{1}{t}}\right)^i. \end{aligned}$$

□

To illustrate the previous lemmas, consider a (1,4,2,3)-archive with 10 million documents. At that point, when a new document is archived, from Lemma 12 we can expect that its parity blocks will be pointed to for the first time when the archive reaches 15 million documents, and for the second time when it reaches 22.5 million documents. From Lemma 14, the expected number of archived documents required until there is one pointer to its parity blocks is 13.96 million, and 19.48 million documents until there are two pointers to its parity blocks. The difference from the results of Lemmas 12 and 14 is due to the skewness of the probability distribution. From Lemma 13, the expected number of archived documents required before a chosen parity block of that document is pointed to the first time is 40 million. As the archive gets bigger, it takes an increasingly long time before a document is pointed to, and during that time it can be tampered with without propagation to any other document in the archive.

The increasing intervals before documents get pointed to suggest a strategy for an attacker who wants to destroy a document: the leaping attack. When executing the leaping attack, the attacker moves away from d_k towards the most recent documents in chronological order, making increasingly larger leaps until it reaches documents that have not been pointed to yet. Using the results presented so far in this section, we prove that when the number of pointers per codeword is *small enough*, the leaping attack can erase a document permanently by deleting a sublinear number of newer documents in the archive.

Theorem 16: Suppose that an attacker wants to erase document d_k from a (s, t, e, p)-archive with uniformly random pointers, and that the number of documents archived after d_k is $K \gg k$. If the number of pointers per document is chosen such that $t < \frac{1}{\ln r}$, where r is the largest real root of the polynomial $x^{p+1} - x^p - x^e + 1 - \frac{1}{p}$, then $E[I_{min}(d_k)] \in o(K)$.

Proof: Consult Appendix C. □

The recurrence relation in the proof Theorem 16[11] is not optimal for three reasons. Firstly, it assumes that each new document to destroy during the attack has only one pointer to a document already destroyed. Pointers that collide are advantageous for the attacker because the involved documents can be destroyed by deleting fewer than e blocks from them. Secondly, it assumes that an attacker always targets parity blocks when completing the deletion of a document. If the attacker is lucky, it is possible that a pointer block can be erased because all the older documents that use it have already

[11]See Eq. (12) in Appendix C.

TABLE I
LOWER BOUND FOR THE NUMBER OF POINTERS FROM THEOREM 16 AND LEMMA 17 FOR DIFFERENT CODE RATES $\frac{s}{p}$

		p								
		2	3	4	5	6	7	8	9	10
	1	4	3	3	3	3	3	3	3	3
	2		4	4	4	3	3	3	3	3
	3			6	5	4	4	4	4	4
	4				7	6	5	5	5	4
s	5					8	7	6	6	5
	6						9	8	7	6
	7							11	9	8
	8								12	10
	9									13

been erased. Thirdly, it assumes that the first few pointers to d point to different parity blocks. This becomes increasingly improbable as p increases and e remains fixed, i.e., if we increase the rate of the code. If the first few pointers to parity blocks of d point to the same block b, then the attacker will not erase it and will be able to propagate the attack further toward the most recent archived documents by targeting the other parity blocks of d. To illustrate this, we slightly improve on Theorem 16 by calculating the expected time required until the most advantageous blocks for the attacker are pointed to for archives with $s = 1$.

Lemma 17: Suppose that an attacker wants to erase document d_k from a $(1, t, 1, p)$-archive with random pointers and that the number of documents archived after d_k is $K \gg k$. If the number of pointers per document is chosen such that $t < \frac{1}{\ln r}$, where $r = \frac{1+\sqrt{5-\frac{4}{p}}}{2}$, then $E[I_{min}(d_k)] \in o(K)$.

Table I contains the lower bound for the number of pointers from Theorem 16 and Lemma 17 for different code rates $\frac{s}{p}$. The only difference is for $\frac{s}{p} = \frac{1}{2}$, for which Theorem 16 gives $t_{min} \geq 3$ and Lemma 17 gives $t_{min} \geq 4$. To compare the theoretical bounds, we simulated the damage caused by the leaping attack on $(1, t, 2, 3)$-archives with 10^6 documents and $t \in \{1, 2, 3, 4, 5, 10\}$ pointers per document chosen uniformly at random. The results are shown in Figure 8. On each graph, the curves represent target documents $\{d_1, d_5, d_{10}, d_{50}, d_{100}, d_{500}, d_{1000}\}$, averaged over 100 simulations. The phase transition as the number of pointers increases is obvious from the graphs. When reaching the threshold, the asymptotic cost of the leaping attack no longer depends on the target document: an attacker who wants to irrecoverably destroy a document must destroy a constant fraction of all documents archived after it. For $\frac{s}{p} = \frac{1}{3}$, the bound given by Theorem 16 and Lemma 17 is $t = 3$, which appears tight when observing Figure 8. Increasing t further accelerates the convergence and increases the fraction of documents that must be destroyed.

A. Optimal Attack and Random Entanglement

We conjecture, with entanglement chosen uniformly at random, that there is a constant number of pointers threshold after which even the optimal attack will require the erasure of a constant fraction of all documents archived after an old enough target. Since simulating the optimal attack is computationally intractable and we do not have good enough theoretical lower bounds, to support this conjecture we simulate the bounded breadth-first search attack described in Section VI-C. By bounding the size of the buffer, we can control the number of nodes traversed at each level of the solution tree.

We use the bounded breadth-first search algorithm with two heuristics: the minimum (Section VI-A.1) and tailored (Section VI-A.4) heuristics. The tailored heuristic is based on Lemma 14: when we decide whether or not to include a new document in the attacked set, we estimate the beginning of its propagation to other documents with Lemma 14 times the number of blocks to erase in the document. This is more accurate than selecting the document that propagates to the smallest number of documents (minimum heuristic), or selecting the document leaping as far as possible towards the end of the archive (leaping heuristic).

Figure 9 shows the damage caused by the tailored bounded breadth-first search attack on $(1, 5, 2, 3)$-archives of size 10^4 with pointers chosen uniformly at random and target document d_i for $i \in \{1, 5, 10, 50, 100\}$. Each curve represents a different tree width (buffer size). The leaping attack is also shown for comparison. Each curve is the average over 100 simulations. Figure 10 shows the damage caused by the minimum bounded breadth-first attack with the same simulation parameters. The figures show the efficiency of the greedy leaping attack. Furthermore, they show that the number of documents that must be erased to compromise a target converges to a constant fraction of the archive, as we conjectured for the optimal attack.

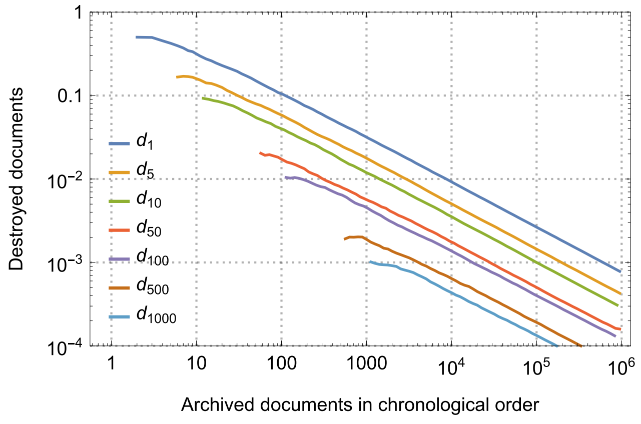
IX. EXTENSIONS AND DISCUSSION

In this section, we discuss in greater detail the assumptions made and constraints self-imposed in this article. We also extend and relax them in various ways, and examine the consequences.

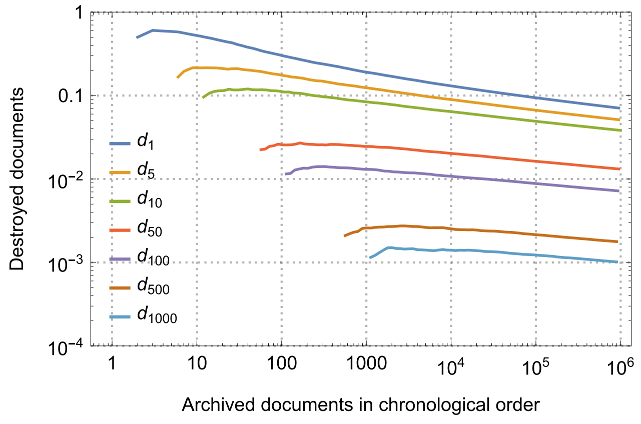
A. To Store or Not to Store Source Blocks?

In the presented architecture, we use codes in semi-systematic form and do store the source blocks. This comes at a non-negligible performance cost: accessing a document requires decoding of its corresponding codeword, and thus fetching the required threshold of valid blocks. Even with a fast decoding algorithm, we still incur the bandwidth overhead of fetching $s + t$ blocks to retrieve s effective blocks of data.

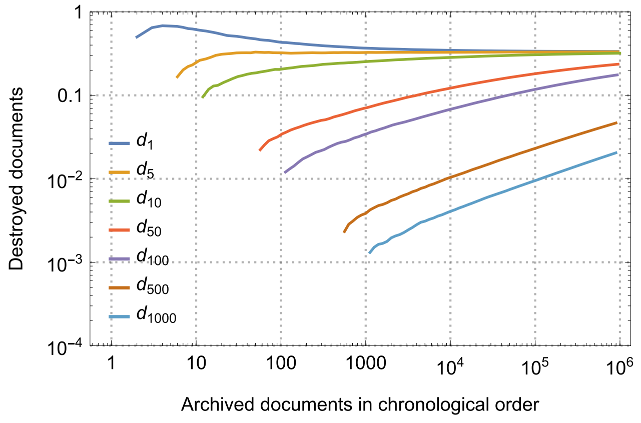
It is tempting to remove this overhead by including the source blocks in the systematic form of the code, and allow the clients to directly fetch the source blocks. However, from the censorship-resistance perspective, a systematic code would invalidate all our previous reasoning. We can no longer equate recoverability of a codeword and recoverability of the corresponding document, as the source blocks may still be recoverable even though the codeword is below the decoding threshold. In an extreme case, the adversary can censor all the stored parity blocks, losing all the redundancy in the storage system, without degradation of service. Targeted source blocks can then be destroyed at will, making their corresponding codewords undecodable; all more recent codewords having



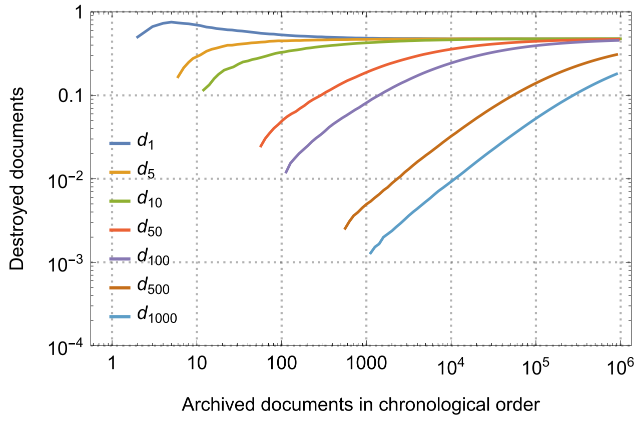
(a)



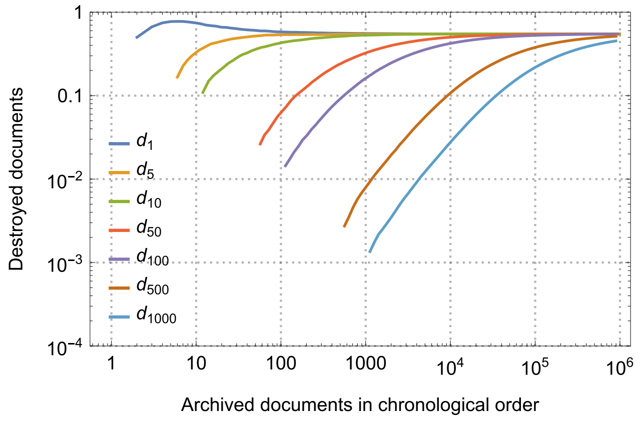
(b)



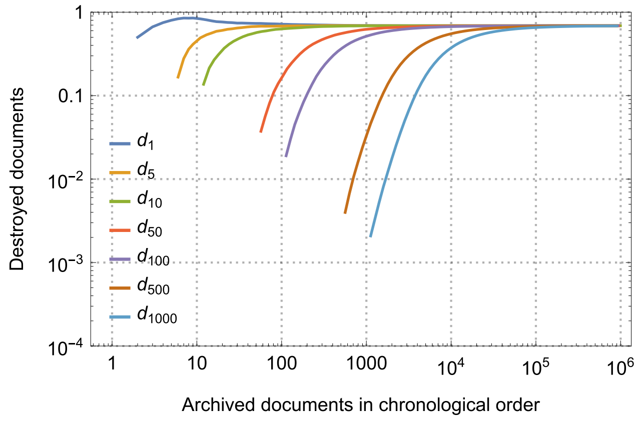
(c)



(d)



(e)



(f)

Fig. 8. Damage caused by the leaping attack on $(1, t, 2, 3)$-archives of size 10^6 with pointers chosen uniformly at random and $t \in \{1, 2, 3, 4, 5, 10\}$. On each graph, the curves represent target document $\{d_1, d_5, d_{10}, d_{50}, d_{100}, d_{500}, d_{1000}\}$, respectively. Each curve is the average over 100 simulations. (a) $t = 1$ pointer. (b) $t = 2$ pointers. (c) $t = 3$ pointers. (d) $t = 4$ pointers. (e) $t = 5$ pointers. (f) $t = 10$ pointers.

pointers to these source blocks immediately become undecodable if they weren't already so, and the attack does not need to propagate further. A system with stored and readily available source blocks thus provides no censorship-resistance, as this attack, even at a high material cost, does not affect data availability.

This material cost can further be reduced. An adversary could tweak his attack algorithm to always spare the source blocks (except for the original target). This results in a system that has lost redundancy for some documents but is still able to directly serve all non-censored source blocks. Recall that for all documents but the initial target, the attacker needs to destroy at most e blocks (the code requires $e + 1$ errors to fail, but at least one block is already missing, otherwise the document would not have entered the target set). We have $e \leq p - s$ (with equality reached for MDS codes), so it is always possible to exclude the source blocks from the attack, and choose among the $p - s$ remaining blocks. This offers a complexity tradeoff between computational effort versus number of blocks to make the attack irrecoverable.

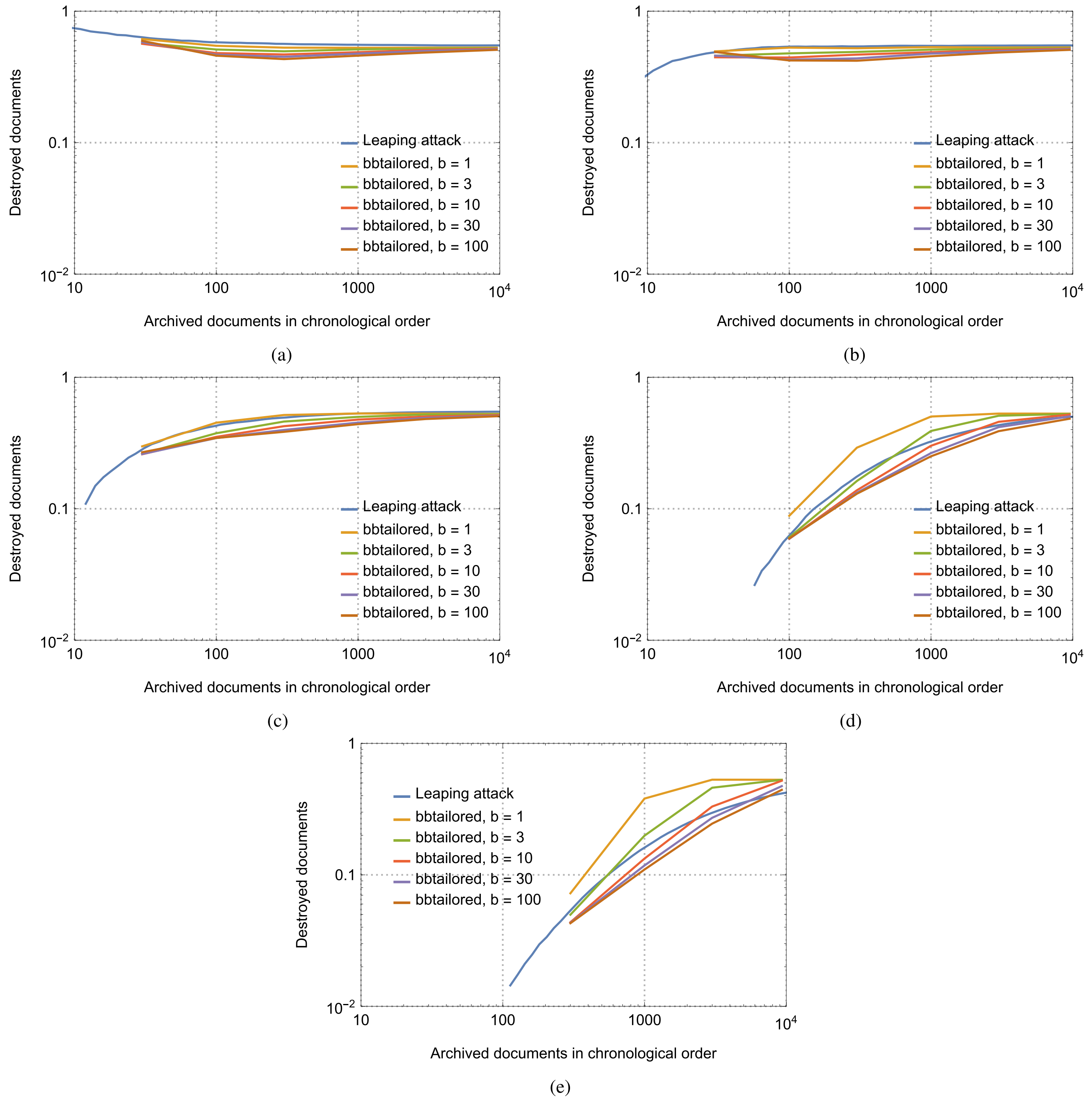


Fig. 9. Damage caused by the tailored bounded breadth-first search attack on (1, 5, 2, 3)-archives of size 10^4 with pointers chosen uniformly at random and target document d_i for $i \in \{1, 5, 10, 50, 100\}$. Each curve represents a different tree width (buffer size). The leaping attack is also shown for comparison. Each curve is the average over 100 simulations. (a) Target document d_1. (b) Target document d_5. (c) Target document d_{10}. (d) Target document d_{50}. (e) Target document d_{100}.

One can wonder whether we can prevent this by hiding which of the p parity blocks are actual source blocks. This could be achieved by storing the identity of the source blocks separately from the rest of the metadata. Users with access to the public metadata could still repair the system, without needing to know the identity of the source blocks. This is akin to the *Repair Capabilities* mechanism of Tahoe [8]. However, if clients use this fast path to access documents, a passive adversary monitoring access to the system could very quickly deduce which blocks are the source blocks. To counter this, clients could request all the blocks and avoid decoding, but this does not appear a worthwhile tradeoff considering the bandwidth overhead.

B. Hiding Metadata

We assumed so far that the metadata, providing the associations between blocks and documents (and required for decoding) would be made public. Our attacks have relied on this metadata being public, but so do our repair algorithms.

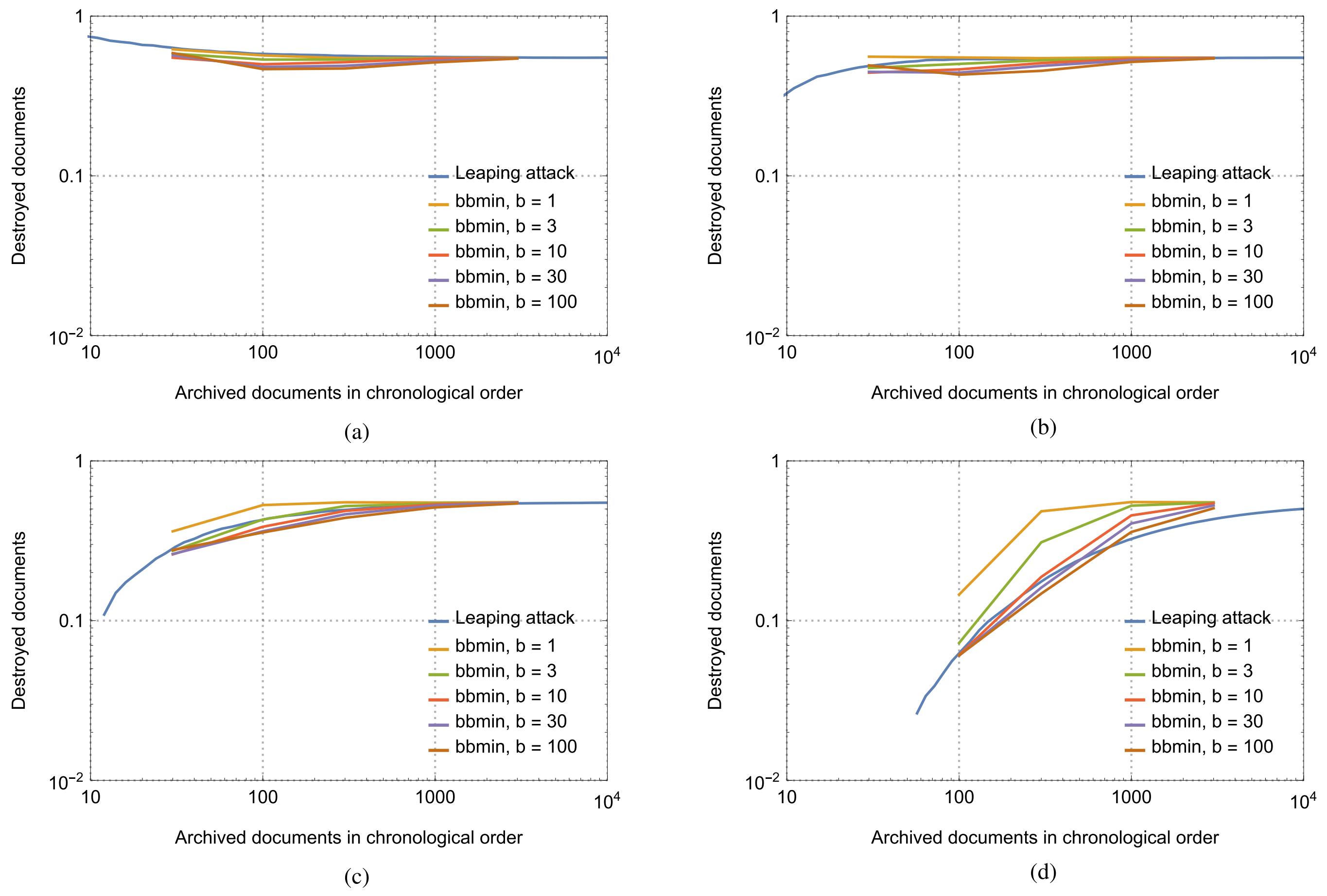


Fig. 10. Damage caused by the minimum bounded breadth-first search attack on (1, 5, 2, 3)-archives of size 10^4 with pointers chosen uniformly at random and target document d_i for $i \in \{1, 5, 10, 50\}$. Each curve represents a different tree width (buffer size). The leaping attack is also shown for comparison. Each curve is the average over 100 simulations. (a) Target document d_1. (b) Target document d_5. (c) Target document d_{10}. (d) Target document d_{50}.

One can wonder whether hiding the metadata from the public results in a net gain or loss of censorship resistance for the users.

In our model, a block can be in only two states, directly recoverable (meaning the system is able to provide an uncorrupted version of the block when queried for it), or lost. The block could be considered lost for several reasons, including that the actual storage has been silently corrupted, that the data is missing, or even that the data is intact and present, but that the system has been coerced into ignoring requests for such a block. The latter scenario is relevant in a censorship context, where it might be easier for a censor to impose a blacklist of forbidden content than to endlessly hunt and remove blocks as they pop up on various systems.

Consider a system where the metadata is kept privately. In such a context, the adversary has no *a priori* information about the relationships between blocks, but has been made aware of the existence of a document to censor by revelation of the corresponding metadata. In such a model, if enough blocks are lost to make a document irrecoverable, users must wait for a third party to access an entangled document. The said third party must then notice that not all blocks are healthy, and after decoding the document, proceed to reupload the missing blocks at its own expense. Some techniques, like the mechanism of repair caps in Tahoe [8], can delegate repairs to incentivized third parties, however to do so they must be trustworthy enough to receive a copy of the secret metadata. Furthermore, a system with private metadata is in position of censoring repair attempts simply by refusing to serve the forbidden blocks, no matter how many third parties try to reupload them. By contrast, if the metadata is public, users can perform their own repairs as long as the attack is not irrecoverable, and for this they do not need an additional incentive. Reuploading the missing content is still at their own cost, but the system has a direct incentive to accept it: every missing block costs the system up to $s + t$ fetches of other blocks. In fact, it saves bandwidth for the system to perform the repair itself.

C. Hiding Timing Information

The proposed basic attacks all heuristically rely on the attacker being able to infer a strict ordering of documents in time. On the other hand, the reconstruction algorithm does not need such information. This leads to two important questions: is it possible to hide this timing information from an attacker, and is it useful?

Suppose the metadata has been stripped of any explicit timestamps. Can the attacker still compute an ordering of the documents? With the exception of anchor blocks, every block in the system must be included as a parity block in exactly one document, and may appear any number of times as a pointer block in newer documents. This property is sufficient to define a partial order on the documents.

An adversary can use a topological sort algorithm to generate a compatible total order in linear time [41]. Because block relationships are the only thing that matter to our attack algorithms, such a total order would be suitable to run the attack. Therefore, if we want to gain something by hiding the document order, we have to ensure that the adversary cannot deduce the role (stored or pointer) of blocks in a particular codeword.

D. Hiding Block Roles

So far, all our representations of the system used a code which had a fixed mapping from code slots to roles. We can randomize this mapping without loss of generality. In a typical setting, our initial, systematic, (n, k) MDS code, the encoding process is used to generate codewords $c_{k+1}, \ldots, c_n$ from the k source words $c_1, \ldots, c_k$, whereas the decoding process can find a unique solution for all the c_i, from any k-subset of them. As such, encoding is a special case of decoding (possibly with better performance, depending on the code).

Therefore, when generating a new document, it is not mandatory to have a fixed mapping between roles and slots. The client can randomly allocate the pointer blocks to slots, then compute the missing entries. Then, the adversary can no longer *locally* distinguish between pointers and parity blocks of a document.

Unfortunately, even if the system stores the metadata in such a way that the block roles are locally obscured, an attacker with a global view of the metadata can always recover such roles with the following algorithm. The adversary can look for blocks that appear exactly once in the whole system, and will know that such occurrences are necessarily parity blocks. Because the set of documents is finite and not empty, and our block role property is transitive (a requirement for a partial order), there must be at least one maximal document whose parity blocks are not entangled with other documents. For these maximal documents, the adversary will be able to infer the role of all the parity blocks. If we assume that the amount of parity and pointer blocks per document is a global constant, the attacker knows he has successfully identified all the parity occurrences of these maximal documents. All other occurrences of blocks can be identified as pointers. The attacker can then remove these maximal documents (which will sit at the top of the total order), and repeat the process recursively while ignoring occurrences of blocks in the removed documents. Eventually, the set of documents ends up empty with all the occurrence roles known.

E. Metadata Write Access and Recoding Attack

We assumed that an attacker can read, but cannot alter the archive metadata. The underlying assumption is that metadata are small enough to be mass replicated using a blockchain or stored locally by the clients interested in a document. Recall that destroying all copies of the metadata would make a document impossible to access, but that this is not worse than losing the keys of the upper encryption layer.

Definition 18: Consider a (s, t, e, p)-archive. An extended integrity set *$E(d_k)$ is a set of documents containing d_k such that it is possible to erase at least $e + s + 1$ blocks per document in $E(d_k) \setminus \{d_k\}$, at least $e + 1$ blocks in document d_k, and no block in any document in the complementary set $\overline{E(d_k)}$. We write $E_{min}(d_k)$ to denote the size of the smallest extended integrity set of document d_k.*

If an attacker can write metadata, it can, instead of destroying files, decode and recode documents without destroying them, and rewrite metadata accordingly. When doing such a *recoding attack*, the attacker first decodes the document d_k to censor, replaces its content (source blocks) with something else, erases $e + 1$ other blocks from the document codeword, and recodes the censored document back in the archive. Changing the target codeword will also affect documents pointing to its erased blocks, thus the attacker must recursively recode the documents pointing to the censored document and erase enough blocks in the original codewords to allow recoding. The difference between integrity sets and extended integrity sets is that the source blocks of a recoded document d_k are recoverable, thus to make the original codeword of d_k irrecoverable, the attacker must erase at least $e+s+1 = p+1$ of its blocks instead of $e + 1$.

Theorem 19: The size $|S|$ of the smallest irrecoverable recoding attack of a target document d_k is equal to $E_{min}(d_k)$.

Proof: We first prove that $|S| \le E_{\min}(d_k)$. From the smallest extended integrity set $E_{\min}(d_k)$, an attacker can decode all the documents in it, erase all the blocks required to realize the extended integrity set, recode a censored version of d_k, and recode all the other documents $d_i \in E_{\min}(d_k) \setminus \{d_k\}$. The original codewords of documents $d_i \in E_{\min}(d_k) \setminus \{d_k\}$ are missing at least $e + s + 1$ blocks, thus even by decoding the new codewords and recovering the source blocks, the old codewords will miss at least $e+1$ blocks, thus none of them is recoverable. This recoding attack requires $E_{\min}(d_k)$ recodings, thus $|S| \le E_{\min}(d_k)$.

We now show that $|S| \ge E_{\min}(d_k)$. Let S be the smallest recoding attack. The original codeword of d_k has at least $e + 1$ erased blocks, otherwise d_k could be recovered. Next, consider a document $d_i \in S$ such that $d_i \neq d_k$. We argue that the original codeword of d_k before recoding has at least $e + s + 1$ erased blocks. Suppose that it is false. Using the source blocks of the recoded documents, we run the reconstruction algorithm and recover a nonempty set O of original codewords including the original codeword of d_i. If C includes the original codeword of d_k, then d_k can be recoded, which is a contradiction, whereas if C does not include it, it follows that all the codewords in C did not have to be recoded, which contradicts the fact that S is the smallest recoding attack. Hence, S is an extended integrity set and $E_{\min}(d_k) \le |S|$. □

The size difference between the optimal recoding attack and the optimal attack can be arbitrarily large. Consider a (1,1,2,3)-archive where the pointer block from document d_i is entangled with one of the parity blocks of d_{i-1} for all i. We can easily show that $I_{\min}(d_k) = 2$ and $E_{\min}(d_k) = N+1$, where $N > 0$ is the number of documents archived after d_k. Thus, an attacker can tamper with d_k irrecoverably by also erasing d_{k+1}, but if it does not want to destroy other documents, even with complete control over the data and metadata, it must recode the N documents archived after d_k.

F. STEP-*Archival Versus WORM Storage*

There have been proposals for immutable storage systems with hardware enforcement. In such a system, the storage controller (or the medium itself) will refuse or fail to honor write requests that would overwrite existing data. It is assumed that a hostile party (a rogue administrator, or an outside attacker) may be able to compromise the operating system, but not the firmware of the storage controller (or the laws of physics, in the case of write-once media). In fact, WORM technologies do not guarantee data retention when the insider can physically access the storage media.

Furthermore, these techniques are applicable to fundamentally different scenarios than the ones we are considering. The storage controller can only guarantee that, as long as the storage device is operating properly, all data ever written is available for reading by the main system. A malicious administrator could still practice censorship by having the main system refuse to perform read requests for censored data. In such a system, it would still be possible to restore access to the censored data by taking back control of the main system.

When facing censorship from a legal authority, however, it may not be possible to legally “take back control” and restore the original intended system behavior. Again, with our approach, it does not matter if censorship is implemented by physically destroying blocks or by making them unavailable for reading through other means. As long as decoding and metadata management is performed on the client, the storage system would need to either guess the intent of the client, and selectively prevent access (and only achieve probabilistic censorship), or prevent access without regard for the decoding intent (and cause collateral damage).

X. CONCLUSION AND OPEN PROBLEMS

In this article, we introduced and studied STEP-archives with the objective of providing tampering and censorship resistance for long-term data storage and permanent archiving in a practically implementable manner. Our long-term goals are a proof of concept and a large-scale implementation, and to achieve those goals there are theoretical and practical questions to explore.

A. Variable Block Size

Imposing a fixed block size on the system level is rigid and inefficient. An unnecessarily large block size wastes storage space, decoding time and bandwidth, but a too small block size bloats metadata and increases the number of I/O operations. Because we need to be able to choose pointers to other blocks, allowing arbitrary block sizes would considerably complicate the pointer selection process. A reasonable compromise is to allow a limited selection of regular block sizes, for instance powers of a power of 2. Furthermore, one could split bigger blocks or concatenate smaller blocks when choosing pointers. This, however, considerably complicates the analysis of the system behaviour under attack as one must also consider partially erased blocks. It also potentially introduces additional overhead, since not all storage APIs allow to fetch partial blocks.

B. Data Expiration

We have no procedure to delete obsolete data and reclaim storage space. While the impossibility to delete old data is paramount to obtain strong censorship resistance, and while storage technologies have shown exponential improvements ever since the advent of computing, we still wish to explore ways to relax this constraint. One might try to extend our framework to only allow the owner to delete data, but this is problematic in many ways. First, in a deduplicated system, there might be more than one rightful owner. Allowing a single owner to trigger the deletion would effectively let him censor the other owners. Therefore, we would need a consensus mechanism to ensure all owners effectively wish to delete. Second, allowing the owner to drive deletion would make him a designated target for a determined censor. Third, deletion of data by its owner, for instance a bank tampering with its transaction records, is not always a desirable feature. We can avoid that simply by not giving the owner, or original creator, any special privileges regarding preservation of data.

Still, it might be desirable for a system to be able to reclaim storage from old, stale data. Unbounded pointer distributions, like the uniform distribution we used in section VIII, do not allow this. But in principle, if bounded distributions like the sliding windows used in section VII are used, it would be possible for a system to remove old data progressively, providing a bound on the age of data recoverable by the clients.

In such a system, uploaded content would have an expiration date, and clients would be responsible for periodically re-inserting the data they do not want to see disappear when it expires. Because this re-insertion step would require coding against a different set of blocks, clients would no longer be able to rely on the metadata hash to authenticate a document.[12] The system would require a more sophisticated mechanism for the users to authenticate the new versions of the metadata blocks. Two possible approaches for this could be either some kind of reputation system vetting on the authenticity of the new metadata, or a proof of storage [26], [29] mechanism allowing a client to interactively query the storage system, to ensure with high probability the legitimacy of a new metadata block.

C. Entanglement and Coding

The main theoretical open problem is to derive non-trivial lower bounds on the cost of the optimal irrecoverable attack. Intuitively, the leaping attack is a good approach against uniform random entanglement, and we conjecture that the optimal attack will not fare better when the number of pointers passes

[12]Even if a client had the hashes of the relevant source blocks, he would be forced to pay the t bandwidth cost and proceed with the decoding before being able to check it.

a threshold since no amount of backtracking will compensate for the exponential propagation of the dependencies in the archive.

Another challenge to overcome is to provide quick tamper resistance after archiving new data while providing strong censorship-resistance in the long term. Essentially, we want the benefits of regular entanglement in a sliding window and the benefits of uniformly random entanglement. A prototype in this direction involves pointers selected using a half-normal distribution for quick forward integrity, combined with pointers chosen uniformly at random for good long-term protection [42]. The prototype also provides temporary replication for new documents and gets rid of the replicas as they age.

Finally, while we focused on MDS codes in this article, the model can be used with any code allowing encoding of the entangled blocks in systematic form, we want to study how to provide strong censorship-resistance using modern and efficient erasure codes for distributed storage. Furthermore, considering that the bandwidth overhead is proportional to the number of entangled blocks per document, we are especially interested in adapting locally repairable codes to deal with hardware failures.

D. Modeling Malicious Clients and Servers

In our mathematical analysis, we assume that the archive is properly constructed, that is, all metadata is sound and accessible to any client. Furthermore, we assume that all documents in the archive are important. We now discuss what can happen when attackers maliciously construct the archive in ways that break these assumptions. These attacks can be perpetrated by malicious clients or servers alike, but note that for an attacker to behave as such, it must proactively participate in the construction of the archive *before* it knows what to censor or tamper with, hoping that when the time comes, the quality of entanglement will be weaker and facilitate the destruction of the targeted data.

A first possible attack is to store a large number of bogus documents. Later, when performing censorship of targeted data, the attacker can try to direct collateral damage to degrade the bogus documents. This does not reduce the amount of work required to destroy a document (it might even increase it), but decreases the amount of collateral damage to other valid documents. Note, however, that this would not affect systems offering all-or-nothing forward integrity. The attacker can go further and entangle bogus documents in a poor fashion, for instance by selecting all the pointers from documents in the recent past. This creates zones of weak entanglement in the archive, which makes censorship of documents pointing to these zones easier to accomplish. As of this writing more work is required to analyze the repercussions of this attack based on the proportion and location of bogus documents in the archive.

A second attack is to store unsound metadata. Two variants are possible: storing invalid metadata that does not match source blocks, and storing valid metadata for stored blocks that do not form valid codewords. In both cases, the system may appear to allow recovery of some blocks through the corresponding metadata entry, while said recovery would actually be impossible; a legitimate client would only notice the problem after attempting to decode a missing block. This is an important issue, as it can mislead clients about their actual capacity to recover lost blocks. As we build our system on top of a self-integrity verifying store, invalid metadata cannot cause a client to decode invalid data; the malicious metadata would be immediately detected. Valid metadata for invalid codewords can also be detected but requires decoding: a client can fetch all the blocks from a given codeword, erase e of them, decode the erased blocks and verify if they match.

Because metadata soundness can be verified by any third party observer (at the cost of some bandwidth and processing for decoding), a large amount of unsound metadata would eventually be detected. Unsound metadata, when detected, can simply be regarded as nonexistent by the client. Like for bogus documents, creating unsound metadata cannot weaken the integrity of preexisting documents a posteriori, but may weaken it for ulterior documents. Attackers might go further and store invalid codewords followed by a large number of properly coded but bogus documents pointing to the invalid codewords. An honest client could still perform a recursive verification of the metadata before encoding its document, but the bandwidth cost and decoding time would be prohibitive. A shallow check with constant cost may be sufficient to amortize and distribute the verification cost over all clients, but again this likely depends on the proportion and location of invalid codewords and bogus documents in the system, and needs to be investigated further.

Appendix A
Proofs From Section V

Proof of Lemma 6: Let $G = (V, E)$ be a multigraph with loops. We consider the size g_2 of the minimum subgraph of minimum degree at least two that includes a target vertex d_k. We can find the 2-girth over the multigraph by repeating the algorithm for all vertices. Furthermore, although we omit the details, the vertices of the smallest subgraph can be found by keeping track of the vertices in optimal paths during the execution of the algorithm.

The five types of minimum subgraphs of minimum degree at least 2 in a multigraph with loops are shown in Figure 11. If the multigraph has two loops at d_k (Fig. 11a), then $g_2 = 1$. If the multigraph has fewer than two loops at d_k but parallel edges from d_k (Fig. 11b), then $g_2 = 2$. If none of the first two cases applies, then $g_2 = \min(g_2^c, g_2^d, g_2^e)$, where g_2^c is the smallest cycle including d_k (Fig. 11c), which can be found in polynomial time using dynamic programming, g_2^d is the smallest simple path including d_k in the middle and ending at both sides with a cycle, a loop, or parallel edges (Fig. 11d), and g_2^e is the smallest path such that d_k has a loop on one side of a simple path ending at the other side with a cycle, a loop or parallel edges (Fig. 11e).

We now present Algorithm 8, a polynomial-time algorithm to calculate $\min(g_2^d, g_2^e)$ when some vertices can be traversed more than once. The algorithm calculates the smallest path starting from each vertex incident to d_k that ends with a

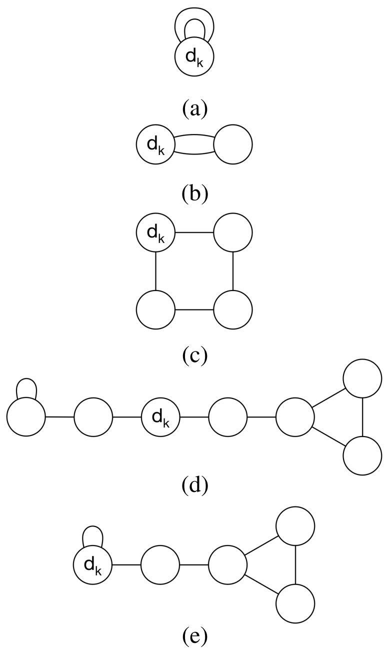


Fig. 11. Types of minimum subgraphs of minimum degree at least 2 in a multigraph with loops. (a) 2 loops at d_k. (b) Parallel edges from d_k. (c) d_k part of a cycle. (d) d_k in the middle of a simple path ending at both sides with a cycle, a loop or parallel edges. (e) d_k with a loop on one side of a simple path ending at the other side with a cycle, a loop or parallel edges.

Algorithm 8: MSMD$_2$ With Repeated Vertices (Types 11d and 11e)

```
input : $V_i(d_k) \leftarrow$ the set of incident vertices to target vertex $d_k$   // $d_k \notin V_i(d_k)$ even if $d_k$ has a loop
output: $\min(g_2^d, g_2^e)$
1 forall the $v_i \in V_i(d_k)$ do
2     forall the $v \in V \setminus \{d_k \cup v_i\}$ do
3         $p(v_i, v) \leftarrow$ length of shortest path from $v_i$ to $v$
4     $p(v_i, v_i) \leftarrow 0$
5 forall the $v \in V \setminus \{d_k\}$ do
6     $c(v) \leftarrow$ length of shortest cycle including $v$
          // The shortest cycle can be a loop or parallel edges if present
7 if there is no loop at $d_k$ then
8     return $\min_{v_{i_1}, v_{i_2} \in V_i,\ v_{i_1} \neq v_{i_2},\ v_1, v_2 \in V \setminus \{d_k\}} (p(v_{i_1}, v_1) + c(v_1) + p(v_{i_2}, v_2) + c(v_2) + 1)$
9 else
10    return $\min_{v_{i_1} \in V_i,\ v_1 \in V \setminus \{d_k\}} (p(v_{i_1}, v_1) + c(v_1) + 1)$
```

cycle, a loop, or parallel edges, and does not pass through d_k. It returns the length of the smallest such path if d_k has a loop (Type 11e), and the sum of the two smallest such paths if d_k is in the middle of the path (Type 11d). For Type 11d, the algorithm does not verify that the two smallest paths are disjoint. However, if they are not disjoint, it implies that d_k is part of a cycle smaller than g_2^e, a case that is accounted for when calculating the smallest cycle including d_k (Type 11c). We conclude the proof by pointing out that all the steps of Algorithm 8 are executed in polynomial time. □

Proof of Theorem 8: Let $G = (V, E)$ be a graph with n vertices all of degree $e+1$ or $e+2$, and whose $e+1$-girth is δ. We prove the theorem by reducing the smallest integrity set problem to finding the $(e+1)$-girth of G, which is **NP**−hard and hard to approximate in polynomial time if $e \geq 2$ [37], [38]. To be precise, we reduce our problem to the $(e+1)$-girth problem that includes a vertex $v_1 \in V$. This problem is as hard as the original $e+1$-girth, otherwise we could solve it n times for the n vertices and solve the original problem. From the constraints of the graph, we can bound the $e+1$-girth by $e+2 \leq \delta \leq n$. This graph has at most $\frac{n(e+2)}{2} \leq ne$ edges. We represent G using an incidence matrix $M_{n\times ne}$, where each row identifies with a vertex, each column with an edge, and $m_{ij} = 1$ if vertex v_i and edge e_j are incident, and 0 otherwise. Each row of M has either $e+1$ or $e+2$ ones.

We construct a (s, t, e, p)−archive A from M, represented as the incidence matrix of its underlying hypergraph. Figure 12 shows the incidence matrix of the top of the archive. Each row represents a document (vertex), and each column a block (hyperedge). Element $a_{ij} = 1$ if block b_j belongs to document d_i, and 0 otherwise. The archive has p archived blocks per document, and t bootstrap anchor blocks. The top of the archive has $n(e+1)$ documents. For each of the first ne documents, the t entangled blocks point to the t anchors (the big green block $1_{ne\times t}$ at the top-left of the figure). For documents d_{ne+1} to $d_{n(e+1)}$, we split the matrix M in columns $m_1, m_2, \ldots, m_{ne}$, and use m_i as part of the hyperedge of the first parity block of d_i for $i \in \{1, 2, \ldots, ne\}$ (blue vertical rectangles in Figure 12). Since we need t pointers per document, we add $t-e-2$ pointers to the first $t-e-2$ anchors for documents d_{ne+1} to $d_{n(e+1)}$, which explains our assumption that $t \geq e+2$ (the green block $1_{ne\times(t-e-2)}$ at the bottom-left of the figure). Since the degree of the vertices of G is $e+1$ or $e+2$, we add a pointer to the last anchor block for document d_{ne+i} if $v_i \in V$ has degree $e+1$ (pink block in the figure). This ensures that each document has exactly t pointers.

Figure 13 shows the incidence matrix of the entire archive, which we extend by adding $\lceil \frac{np}{t} \rceil + 1$ blocks $1_{n(e+1)\times t}$ of pointers and a sufficient number of documents to cover all such blocks. The first such block points to the t anchor blocks, and we cascade the other $\lceil \frac{np}{t} \rceil$ blocks under the parity blocks of documents d_{ne+1} to $d_{n(e+1)}$. The number of archived documents is therefore

$$n(e+1) \cdot \left(\left\lceil \frac{np}{t} \right\rceil + 2 \right) \in \mathcal{O}(n^2).$$

We now explain the motivation behind all these gadgets. We want to find $I_{\min}(d_k)$ where $d_k \triangleq d_{ne+1}$. We can erase all the parity blocks from documents d_1 to d_{ne}. This corresponds to the *integrity region* inside the thick border in Figure 13. It is clear that we erased at least $e+1$ blocks per document from d_1 to $d_{n(e+1)}$, and no block in the other documents. Hence, $I_{\min}(d_k) \leq n(e+1)$. By construction, all the blocks

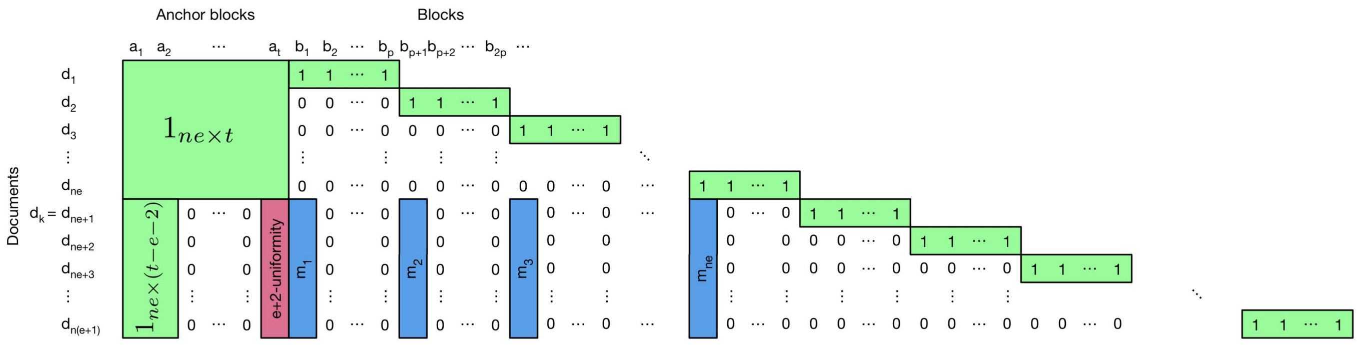


Fig. 12. Polynomial reduction from $I_{\min}(d_k)$ to the $e+1$-girth problem: incidence matrix for the top of the archive.

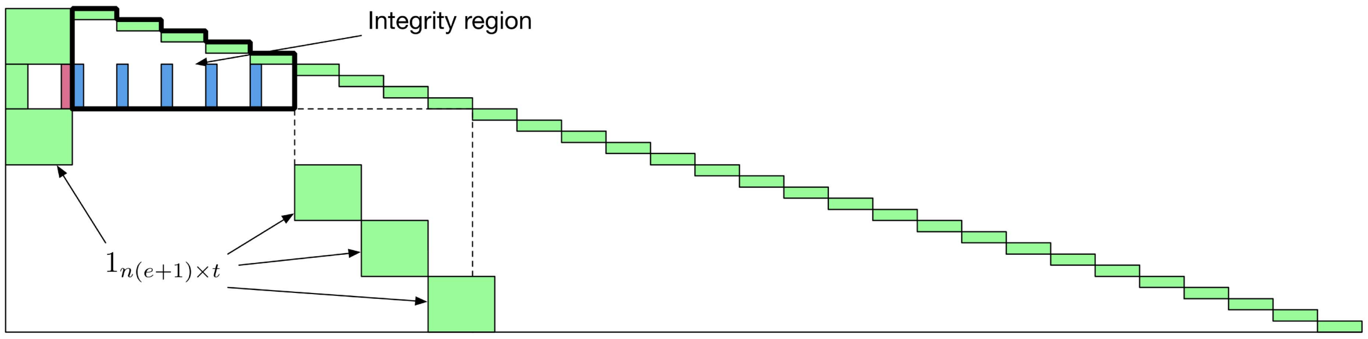


Fig. 13. Polynomial reduction from $I_{\min}(d_k)$ to the $e+1$-girth problem: incidence matrix for the entire archive.

outside the integrity region are used in at least $n(e+1)+1$ documents, thus none of them can be part of the smallest integrity set containing d_k. We can therefore limit our search for the smallest integrity set inside the integrity region.

We now reduce $I_{\min}(d_k)$ to the $(e+1)$-girth of G. Assume without loss of generality that there is exactly one smallest subgraph H, of size δ, of minimum degree at least $e+1$ in G. This minimum subgraph H has at most $\frac{\delta\cdot(e+2)}{2}$ edges. By construction of the archive, it follows that we can form an integrity set by erasing the blocks corresponding to edges in the minimum subgraph. This erases one document per vertex in H (d_{ne+i} is erased if and only v_i is in H), and also erases one document per edge in H (d_i is erased if and only e_i is in H). Thus,

$$\begin{aligned} I_{\min}(d_k) &\leq \delta + \frac{\delta\cdot(e+2)}{2} \\ &\leq \delta\left(1+\frac{e+2}{2}\right). \end{aligned} \tag{1}$$

It is not possible for an integrity set of d_k to contain fewer than δ documents between d_{ne+1} and $d_{n(e+1)}$, because it would imply that the $(e+1)$-girth of G is smaller than δ. The smallest subgraph H has at least $\frac{\delta(e+1)}{2}$ edges. By construction, it means that the smallest integrity set must include at least $\frac{\delta(e+1)}{2}$ blocks in $\delta+\frac{\delta(e+1)}{2}$ documents (one erased document for each edge and vertex in H), thus

$$\begin{aligned} I_{\min}(d_k) &\geq \delta + \frac{\delta\cdot(e+1)}{2} \\ &= \delta\left(1+\frac{e+1}{2}\right). \end{aligned} \tag{2}$$

Putting (1) and (2) together and solving for δ, we obtain

$$\frac{I_{\min}(d_k)}{1+\frac{e+2}{2}} \leq \delta \leq \frac{I_{\min}(d_k)}{1+\frac{e+1}{2}}.$$

Hence, if we could calculate $I_{\min}(d_k)$ in polynomial time, we could also approximate δ within a constant factor in polynomial time, which is not possible if **P** $\neq$ **NP**. Thus, calculating $I_{\min}(d_k)$ is **NP**-hard. Furthermore, if we could approximate $I_{\min}(d_k)$ in polynomial time, we could also approximate δ in polynomial time. The approximation hardness results from [37] and [38] therefore also apply to our problem.

We complete the proof by mentioning that finding $I_{\min}(d_k)$ for $1 \leq k < ne+1$ is also hard. The proof uses the same construction, but reduces $I_{\min}(d_k)$ to the d-girth problem such that the smallest subgraph must contain a specific edge. This forces the erasure of the parity block of d_k that contains the edge. Without this condition, if the edge incident to d_k is not in H, then $I_{\min}(d_k) = 1$. □

Appendix B
Proofs From Section VII

Proof of Lemma 10: We first prove the upper bound. If there are fewer than $2w-1$ documents archived after d_k, then the parity blocks from d_k and the documents that follow can be erased. If there are more than $2w-1$ documents archived after d_k, then we erase the parity blocks in documents d_k to d_{k+w-1} and erase the tangled blocks (pointers) in documents d_{k+w} to d_{k+2w-1}. There are at least $p > e$ erased blocks per document, and the code can only correct e block erasures per codeword. From the sliding window, none of the erased blocks from the first w documents are pointed to by a document newer than d_{k+2w-1}, and none of the erased pointers from the last w documents points to documents older than d_k. To complete the proof, we observe that some of the pointers from the last w documents might also point to parity blocks from other documents among the last w, but these parity blocks cannot be at the same time pointed to by documents newer than d_{k+2w-1} because we assumed that every block in the archive is pointed to at most once. Hence, ${}_{\max}\mathcal{I}_{\min} \leq {}_{\max}\mathcal{W}_{\min} \leq 2w$.

For the lower bound, we construct (s, t, e, p)-archives with $s = 1$. Let $b_k^1, b_k^2, \ldots, b_k^p$ be the parity blocks of document d_k. For every k, the entangled blocks of d_k are set to $t_k^i = b_{k-w+i-1}^i$ for $i \in \{1, 2, \ldots, p-1\}$ and $t_k^p = b_{k-1}^p$. We set $b_j^i \triangleq b_a$ when $j < 1$, thus for the first w archived documents, some of the entangled blocks point to an anchor block b_a. It should be clear that every parity block of d_k will eventually be pointed to exactly once after w additional blocks have been archived.

Since the entanglement structure is the same for every document d_k, we can without loss of generality attack d_k by tampering with documents archived after d_k. Thus, to erase d_k, its p parity blocks must be erased. These parity blocks are used in blocks d_{k+1} and $d_{k+w-p+2}, d_{k+w-p+3}, \ldots, d_{k+w-1}, d_{k+w}$, which must be destroyed.

Consider now block $d_{k+w-p+1}$; we show that we achieve the lower bound whether we erase it or not.

If $d_{k+w-p+1}$ is erased, then the entangled block to document d_{k+w-p} can be erased, but the other entangled blocks come from documents older than d_k and must be kept. That leaves $p-1$ parity blocks that must be deleted. The parity blocks of document $d_{k+w-p+1}$ are pointed to by blocks in documents $d_{k+w-p+2}$ and

$$d_{k+w-p+2+w-p+1}, \quad d_{k+w-p+2+w-p+2}, \ldots, \\ d_{k+w-p+2+w-2}, d_{k+w-p+2+w-1}.$$

By taking the first $p-1$ such blocks, the smallest integrity window for d_k must contain a document at least as recent as $d_{k+w-p+2+w-2} = d_{k+2w-p}$.

If $d_{k+w-p+1}$ is not erased, then the pointer from document $d_{k+w-p+2}$ to document $d_{k+w-p+1}$ cannot be erased, but since $d_{k+w-p+2}$ is erased, p of its other blocks must be erased. Its pointer to d_k is already erased, but its other entangled blocks come from documents older than d_k and must be kept. We must thus erase $p-1$ of its parity blocks, thus a document as least as new as $d_{k+w-p+2+w-1} = d_{k+2w-p+1}$ must be erased.

Hence,

$$W_{\min}(d_k) \geq \min(2w-p+1, 2w-p+2) = 2w-p+1.$$

Since the integrity is the same for every old enough document, we conclude that

$${}_{\max}\mathcal{W}_{\min} \geq \mathcal{W}_{\min}(d_k) \geq 2w-p+1.$$

□

Appendix C
Proofs From Section VIII

Proof of Lemma 12: **Case $i = 1$.** We consider the random variable $L' \geq 1$ defined as

$$L' \triangleq L_1 - k. \tag{3}$$

Since $d_{k+L'}$ is the first document pointing to any of the parity blocks of d_k, all the pointers from documents d_{k+1} to $d_{k+L'-1}$ cannot point to d_k. It follows that

$$\begin{aligned}\Pr[L' = l] &= \frac{\binom{p(k-1)}{t}}{\binom{pk}{t}} \cdot \frac{\binom{pk}{t}}{\binom{p(k+1)}{t}} \cdot \cdots \cdot \frac{\binom{p(k+l-3)}{t}}{\binom{p(k+l-2)}{t}} \\ &\quad \cdot \left(1 - \frac{\binom{p(k+l-2)}{t}}{\binom{p(k+l-1)}{t}}\right) \\ &= \frac{\binom{p(k-1)}{t}}{\binom{p(k+l-2)}{t}} \cdot \left(1 - \frac{\binom{p(k+l-2)}{t}}{\binom{p(k+l-1)}{t}}\right) \\ &= \binom{p(k-1)}{t} \cdot \left(\frac{1}{\binom{p(k+l-2)}{t}} - \frac{1}{\binom{p(k+l-1)}{t}}\right).\end{aligned} \tag{4}$$

The expectation of this random variable is

$$\begin{aligned}E[L'] &= \sum_{l=1}^{\infty} l \cdot \Pr[L' = l] \\ &= \binom{p(k-1)}{t} \sum_{l=1}^{\infty} \left(\frac{l}{\binom{p(k+l-2)}{t}} - \frac{l}{\binom{p(k+l-1)}{t}}\right) \\ &= \binom{p(k-1)}{t} \sum_{l=k-1}^{\infty} \frac{1}{\binom{pl}{t}}.\end{aligned}$$

The series diverges with $t = 1$, whereas when $t > 1$ we can bound $E[L']$ by

$$\begin{aligned}\binom{p(k-1)}{t} \int_{k-1}^{\infty} \frac{1}{\binom{pl}{t}}\, dl &\leq E[L'] \\ &\leq \binom{p(k-1)}{t} \int_{k-1}^{\infty} \frac{1}{\binom{p(l-1)}{t}}\, dl.\end{aligned}$$

Given that $\binom{n}{t} \sim \frac{n^t}{t!}$ for t constant and large n, the upper bound becomes

$$\begin{aligned}E[L'] &\lesssim \frac{p^t(k-1)^t}{t!} \int_{k-1}^{\infty} \frac{t!}{p^t(l-1)^t}\, dl \\ &\leq \frac{k-1}{t-1} \cdot \left(\frac{k-1}{k-2}\right)^{t-1} \\ &\sim \frac{k}{t-1}\end{aligned} \tag{5}$$

and the lower bound becomes

$$\begin{aligned}E[L'] &\gtrsim \frac{p^t(k-1)^t}{t!} \int_{k-1}^{\infty} \frac{1}{\frac{p^t l^t}{t!} + o(l)}\, dl \\ &\gtrsim (k-1)^t \int_{k-1}^{\infty} \frac{1}{l^t + o(l)}\, dl \\ &\gtrsim (k-1)^t \int_{k-1}^{\infty} \frac{1}{(l+1)^t}\, dl \\ &\geq \frac{k-1}{t-1} \cdot \left(\frac{k-1}{k}\right)^{t-1} \\ &\sim \frac{k}{t-1}.\end{aligned} \tag{6}$$

Putting (5) and (6) together it follows that

$$E[L'] \sim \frac{k}{t-1} \tag{7}$$

and from (3), we can conclude that

$$E[L_1] \sim k\left(1+\frac{1}{t-1}\right).$$

Case $i > 1$**.** Since $E[L_1] = \infty$ when $t = 1$, it is clear that $E[L_i] = \infty$ when $t = 1$ and $i > 1$. For $t > 1$, we prove the result by strong induction on i. Since the basis step was done for the case $i = 1$, we can assume that the property is true for $i = 1, 2, \dots, n$ and prove the property for $n + 1$.

Using the iterated expectation, we can write

$$\begin{aligned} E[L_{n+1}] &= E[E[L_{n+1} \mid L_1]] \\ &= \sum_{l_1 \geq k+1} \Pr[L_1 = l_1] \cdot E[L_{n+1} \mid L_1 = l_1]. \end{aligned}$$

The quantity $E[L_{n+1} \mid L_1 = l_1]$ corresponds to the expected position of the $(n+1)$-th document pointing to d_k given that d_{l_1} is the first document pointing to it. Since the pointers are chosen randomly, this is equivalent to the expected position of the n-th document pointing to document d_{l_1}. Since the property is true for $i = n$ from the induction hypothesis, it follows that

$$E[L_{n+1}] = \sum_{l_1 \geq k+1} \Pr[L_1 = l_1] \cdot \left[l_1 \cdot \left(1+\frac{1}{t-1}\right)^n + o(k)\right]. \tag{8}$$

As done in (3) for the case $i = 1$, we use the random variable $L' = L_1 - k$ and rewrite (8) as

$$\begin{aligned} E[L_{n+1}] &= \sum_{l \geq 1} \Pr[L' = l] \cdot \left[(l+k) \cdot \left(1+\frac{1}{t-1}\right)^n + o(k)\right] \\ &= \left(1+\frac{1}{t-1}\right)^n E[L'] \\ &\quad + \left[k\left(1+\frac{1}{t-1}\right)^n + o(k)\right] \sum_{l \geq 1} \Pr[L' = l] \end{aligned} \tag{9}$$

where $\Pr[L' = l]$ is defined as in (**??**). From (7), the first term of (9) can be written as

$$\left(1+\frac{1}{t-1}\right)^n E[L'] = \left(1+\frac{1}{t-1}\right)^n \frac{k}{t-1} + o(k) \tag{10}$$

whereas from (4) the second term of (9) can be written as

$$\begin{aligned} &\left[k\left(1+\frac{1}{t-1}\right)^n + o(k)\right] \sum_{l \geq 1} \Pr[L' = l] \\ &= \left[k\left(1+\frac{1}{t-1}\right)^n + o(k)\right] \cdot \binom{p(k-1)}{t} \\ &\quad \cdot \sum_{l \geq 1} \left(\frac{1}{\binom{p(k+l-2)}{t}} - \frac{1}{\binom{p(k+l-1)}{t}}\right) \\ &= k\left(1+\frac{1}{t-1}\right)^n + o(k). \end{aligned} \tag{11}$$

Putting (10) and (11) together, we can conclude that

$$\begin{aligned} E[L_{n+1}] &= \left(1+\frac{1}{t-1}\right)^n \frac{k}{t-1} + k\left(1+\frac{1}{t-1}\right)^n + o(k) \\ &= k\left(1+\frac{1}{t-1}\right)^n \left(\frac{1}{t-1}+1\right) + o(k) \\ &= k\left(1+\frac{1}{t-1}\right)^{n+1} + o(k) \\ &\sim k\left(1+\frac{1}{t-1}\right)^{n+1}. \end{aligned}$$

□

Proof of Lemma 14: The probability that at least one block of document d_{k+m} points to one of the parity blocks of d_k for $m > 0$ is

$$1 - \frac{\binom{p(k+m-2)}{t}}{\binom{p(k+m-1)}{t}} \sim 1 - \left(\frac{m+k-2}{m+k-1}\right)^t,$$

thus

$$\begin{aligned} E[N_l] &= \sum_{m=1}^{l} \left(1 - \left(\frac{m+k-2}{m+k-1}\right)^t\right) \\ &= l - \sum_{m=1}^{l} \left(\frac{m+k-2}{m+k-1}\right)^t \\ &= l - \sum_{m=1}^{l} \sum_{j=0}^{t} \binom{t}{j} \cdot \frac{(-1)^j}{(m+k-1)^j} \\ &= l - \sum_{m=1}^{l} \left(1 - \frac{t}{m+k-1} + o(k^{-1})\right). \end{aligned}$$

Since by assumption $l \in O(k)$, it follows that

$$\begin{aligned} E[N_l] &= l - l + t \sum_{m=1}^{l} \frac{1}{m+k-1} + o(k) \\ &= t(H_{l+k-1} - H_k) \end{aligned}$$

where H_n is the n-th partial sum of the harmonic series, which can be bounded by $H_n = \ln n + o(n)$ [43]. We can therefore conclude that

$$\begin{aligned} E[N_l] &= t\,(\ln(l+k-1) - \ln(k)) + o(k) \\ &\sim t \ln\left(1+\frac{l}{k}\right). \end{aligned}$$

□

Proof of Theorem 16: We split the leaping attack into levels $1, 2, 3, \dots$ and count, for level n, the number of documents that need to be corrupted between documents $d_{k_{n-1}}$ and d_{k_n}, where

$$k_n = k\left(e^{\frac{1}{t}}\right)^n.$$

We choose this specific k_n because from Lemma 14 and Corollary 15, we can expect that one of the parity blocks from a document in level l will be pointed to once in each subsequent level. Since the number of documents at each level increases by a factor of $e^{\frac{1}{t}}$, the essence of the proof is that the maximum number of documents that need to be corrupted by an attacker at each level grows at a slower rate than $e^{\frac{1}{t}}$ if the number of pointers per document is too small.

Let d be an intermediate document to be erased at level n of the leaping attack. This means that at least one of the pointer blocks of d was already erased when an older document was erased earlier in the attack. Thus, at most e additional block must be further erased from d. It is expected that one document per level greater than n will point to a parity block of d, but since the attacker can choose to erase any e of the p parity blocks, it can skip the first $p-e$ times that a block of d is pointed to, and will propagate the attack to e documents, one in each level from $n+p-e+1$ to $n+p$. Once a block is erased, the probability that it is pointed to by a document at each subsequent level is $\frac{1}{p}$. Let x_n be the expected number of documents that need to be erased at level n. From the previous discussion, x_n can be upper bounded by the recurrence relation

$$x_n = x_{n-p+e-1} + \cdots + x_{n-p} + \frac{1}{p}\sum_{i=1}^{n-p-1} x_i. \qquad (12)$$

To solve this recurrence relation, we can write

$$x_{n-1} = x_{n-p+e-2} + \cdots + x_{n-p-1} + \frac{1}{p}\sum_{i=1}^{n-p-2} x_i, \qquad (13)$$

and by subtracting (13) from (12) it follows that

$$\begin{aligned} x_n - x_{n-1} &= x_{n-p+e-1} + \frac{1}{p}x_{n-p-1} - x_{n-p-1} \\ x_n &= x_{n-1} + x_{n-p+e-1} + \left(\frac{1}{p} - 1\right) x_{n-p-1}. \end{aligned}$$

The solution of the recurrence relation is

$$x_n = C_1\ r_1^n + C_2\ r_2^n + \cdots + C_{p+1} r_{p+1}^n$$

where the C_i are constants and the r_i are the roots of the characteristic polynomial[13]

$$f(x) = x^{p+1} - x^p - x^e + 1 - \frac{1}{p}.$$

The rate of growth of x_n when n is large is

$$\begin{aligned} \lim_{n\to\infty} \frac{x_{n+1}}{x_n} &= \frac{C_1\ r_1^{n+1} + C_2\ r_2^{n+1} + \cdots + C_{p+1} r_{p+1}^{n+1}}{C_1\ r_1^n + C_2\ r_2^n + \cdots + C_{p+1} r_{p+1}^n} \\ &= r \end{aligned}$$

where r is the largest real root of $f(x)$.

If $r < e^{\frac{1}{t}}$, equivalently if $t < \frac{1}{\ln r}$, it follows that the number of documents at each level increases faster than the number of documents that need to be tampered with at each level during the leaping attack, thus $E[I_{\min}(d_k)] \in o(K)$. $\square$

Proof of Lemma 17: Let M be the random variable defined such that $M = m$, for $m \geq 1$, means that the first m pointers to parity blocks of document d point to the same block, and the $m+1$-th pointer points to a different block. It should be clear that an attacker working on the leaping attack will not choose the first document pointed to and that

$$\Pr[M = m] = \left(\frac{p-1}{p}\right)^m.$$

[13]Without loss of generality, we assume that the roots have multiplicity one. The analysis that follows remains valid if the roots are not all distinct.

We use a reasoning similar to the one in the proof of Theorem 16. The recurrence relation for $M = m$ is given by

$$x_n(m) = x_{n-m-1} + \frac{1}{p}\sum_{i=1}^{n-m-2} x_i.$$

The asymptotic recurrence relation for the leaping attack is therefore

$$\begin{aligned} x_n &= \sum_{m=1}^{\infty} \Pr[M = m] \cdot x_n(m) \\ &= \frac{p-1}{p}\sum_{n=1}^{n-2} x_i. \end{aligned} \qquad (14)$$

To solve this recurrence relation, we can write

$$x_{n-1} = \frac{p-1}{p}\sum_{n=1}^{n-3} x_i \qquad (15)$$

and by subtracting (15) from (14) it follows that

$$x_n = x_{n-1} + \frac{p-1}{p} x_{n-2}.$$

The solution of the recurrence relation is

$$x_n = c_1 \left(\frac{1 + \sqrt{5 - \frac{4}{p}}}{2}\right)^n + c_2 \left(\frac{1 - \sqrt{5 - \frac{4}{p}}}{2}\right)^n$$

and rate of growth of x_n when n is large is

$$r \triangleq \lim_{n\to\infty} \frac{x_{n+1}}{x_n} = \frac{1 + \sqrt{5 - \frac{4}{p}}}{2}.$$

Hence, if $t < \frac{1}{\ln r}$, then the number of documents at each level increases faster than the number of documents that need to be tampered with at each level during the leaping attack and we can conclude that $E[I_{\min}(d_k)] \in o(K)$. $\square$

Acknowledgment

The authors would like to thank the anonymous referees for their comments, which greatly improved the positioning and presentation of the work.

References

[1] R. Anderson, "The eternity service," in *Proc. PRAGOCRYPT*, 1996, pp. 242–252.

[2] M. Waldman, A. D. Rubin, and L. F. Cranor, "Publius: A robust, tamper-evident, censorship-resistant Web publishing system," in *Proc. 9th USENIX Secur. Symp.*, 2000, pp. 1–14.

[3] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Designing Privacy Enhancing Technologies*. New York, NY, USA: Springer-Verlag, 2001, pp. 46–66.

[4] R. Dingledine, M. Freedman, and D. Molnar, "The Free Haven project: Distributed anonymous storage service," in *Designing Privacy Enhancing Technologies*. New York, NY, USA: Springer-Verlag, 2001, pp. 67–95.

[5] A. Stubblefield and D. S. Wallach, "Dagster: Censorship-resistant publishing without replication," Dept. Comput. Sci., Rice Univ., Tech. Rep. TR01-380, Jul. 2001.

[6] M. Waldman and D. Mazières, "Tangler: A censorship-resistant publishing system based on document entanglements," in *Proc. 8th ACM Conf. Comput. Commun. Secur.*, 2001, pp. 126–135.

[7] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh, "SiRiUS: Securing remote untrusted storage," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2003, pp. 1–15.

[8] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: The least-authority filesystem," in *Proc. 4th ACM Int. Workshop Storage Secur. Survivability*, 2008, pp. 21–26.

[9] M. Backes, M. Hamerlik, A. Linari, M. Maffei, C. Tryfonopoulos, and G. Weikum, "Anonymous and censorship resistant content sharing in unstructured overlays," in *Proc. 27th ACM Symp. Principles Distrib. Comput.*, New York, NY, USA, 2008, p. 429. [Online]. Available: http://doi.acm.org/10.1145/1400751.1400822

[10] C. Troncoso, G. Danezis, M. Isaakidis, and H. Halpin, "Systematizing decentralization and privacy: Lessons from 15 years of research and deployments," in *Proc. Int. Symp. Privacy Enhancing Technol.*, 2017, pp. 404–426.

[11] G. Perng, M. K. Reiter, and C. Wang, "Censorship resistance revisited," in *Proc. Int. Conf. Inf. Hiding*, 2005, pp. 62–76. [Online]. Available: http://dx.doi.org/10.1007/11558859_6

[12] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Proc. 36th Annu. Symp. Found. Comput. Sci.*, Milwaukee, WI, USA, Oct. 1995, pp. 41–50. [Online]. Available: https://doi.org/10.1109/SFCS.1995.492461

[13] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[14] J. Aspnes, J. Feigenbaum, A. Yampolskiy, and S. Zhong, "Towards a theory of data entanglement," *Theor. Comput. Sci.*, vol. 389, nos. 1–2, pp. 26–43, 2007.

[15] G. Ateniese, Ö. Dagdelen, I. Damgård, and D. Venturi, "Entangled cloud storage," *Future Generat. Comput. Syst.*, vol. 62, pp. 104–118, Sep. 2016. [Online]. Available: https://doi.org/10.1016/j.future.2016.01.008

[16] (May 12, 2003). *SEC Interpretation: Electronic Storage of Broker-Dealer Records. Securities and Exchange Commission 17 CFR Part 241*. [Online]. Available: https://www.sec.gov/rules/interp/34-47806.htm

[17] A. Atabaki, E. Charkes, M. Cordisco, and H. Striplin, "The do's and don'ts of record retention," in *Proc. FINRA Annu. Conf.*, Washington, DC, USA, May 2016. [Online]. Available: http://www.finra.org/sites/default/files/2016_AC_Record_Retention.pdf

[18] R. Sion and Y. Chen, "Fighting Mallory the insider: Strong write-once read-many storage assurances," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 2, pp. 755–764, Apr. 2012. [Online]. Available: https://doi.org/10.1109/TIFS.2011.2172207

[19] *EMC Centera Content Addressable Storage*. Accessed: Apr. 15, 2018. [Online]. Available: http://emc.com/collateral/hardware/data-sheet/c931-emc-centera-cas-ds.pdf

[20] "Best practices to secure data from modification: Eliminating the risk to online content," GreenTec-USA Inc., Sterling, VA, USA, GreenTec WORM White Paper. [Online]. Available: http://greentecusa.com/wp/GreenTec-WORM-Whitepaper.pdf

[21] A. Sparkes and M. J. Spitzer, "Retention management in a WORM storage system," U.S. Patent 9 639 540 B2, May 2, 2017.

[22] *HubStor Compliance Cloud Storage*. Accessed: Apr. 15, 2018. [Online]. Available: http://www.hubstor.net/worm-compliance-cloud-storage

[23] A. Abe *et al.*, "File system implementing write once read many (WORM), " U.S. Patent 9 659 028 B2, May 23, 2017.

[24] A. Abe *et al.*, "File system implementing write once read many (WORM), " U.S. Patent 9 659 029 B2, May 23, 2017.

[25] G. Ateniese *et al.*, "Provable data possession at untrusted stores," in *Proc. ACM Conf. Comput. Commun. Secur.*, Alexandria, VA, USA, 2007, pp. 598–609. [Online]. Available: http://doi.acm.org/10.1145/1315245.1315318

[26] A. Juels and B. S. Kaliski, Jr., "Pors: Proofs of retrievability for large files," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 584–597.

[27] R. Curtmola, O. Khan, R. C. Burns, and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in *Proc. 28th IEEE Int. Conf. Distrib.Comput. Syst. (ICDCS)*, Beijing, China, Jun. 2008, pp. 411–420. [Online]. Available: https://doi.org/10.1109/ICDCS.2008.68

[28] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proc. 4th Int. Conf. Secur. Privacy Commun. Netw.*, Istanbul, Turkey, 2008, Art. no. 9. [Online]. Available: http://doi.acm.org/10.1145/1460877.1460889

[29] Q. Zheng and S. Xu, "Secure and efficient proof of storage with deduplication," in *Proc. 2nd ACM Conf. Data Appl. Secur. Privacy*, New York, NY, USA, 2012, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/2133601.2133603

[30] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," *ACM Trans. Inf. Syst. Secur.*, vol. 17, no. 4, 2015, Art. no. 15. [Online]. Available: http://doi.acm.org/10.1145/2699909

[31] H. Mercier, M. Augier, and A. K. Lenstra, "STEP-archival: Storage integrity and anti-tampering using data entanglement," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2015, pp. 1590–1594. [Online]. Available: https://doi.org/10.1109/ISIT.2015.7282724

[32] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2004.

[33] D. Vorick and L. Champine, *Sia: Simple Decentralized Storage*. Accessed: Apr. 15, 2018. [Online]. Available: https://www.sia.tech/whitepaper.pdf

[34] S. Wilkinson *et al.*, *Storj: A Peer-to-Peer Cloud Storage Network*. Accessed: Apr. 15, 2018. [Online]. Available: https://storj.io/storj.pdf

[35] P. Erdős, R. J. Faudree, C. C. Rousseau, and R. H. Schelp, "Subgraphs of minimal degree k," *Discrete Math.*, vol. 85, no. 1, pp. 53–58, 1990. [Online]. Available: http://dx.doi.org/10.1016/0012-365X(90)90162-B

[36] B. Bollobás and G. Brightwell, "Long cycles in graphs with no subgraphs of minimal degree 3," *Discrete Math.*, vol. 75, nos. 1–3, pp. 47–53, 1989. [Online]. Available: http://dx.doi.org/10.1016/0012-365X(89)90077-0

[37] O. Amini, D. Peleg, S. Pérennes, I. Sau, and S. Saurabh, "On the approximability of some degree-constrained subgraph problems," *Discrete Appl. Math.*, vol. 160, no. 12, pp. 1661–1679, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.dam.2012.03.025

[38] D. Peleg, I. Sau, and M. Shalom, "On approximating the d-girth of a graph," *Discrete Appl. Math.*, vol. 161, nos. 16–17, pp. 2587–2596, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.dam.2013.04.022

[39] O. Amini, I. Sau, and S. Saurabh, "Parameterized complexity of finding small degree-constrained subgraphs," *J. Discrete Algorithms*, vol. 10, pp. 70–83, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jda.2011.05.001

[40] J. Flum and M. Grohe, *Parameterized Complexity Theory* (Texts in Theoretical Computer Science. An EATCS Series). Berlin, Germany: Springer, 2006.

[41] R. E. Tarjan, "Edge-disjoint spanning trees and depth-first search," *Acta Inf.*, vol. 6, no. 2, pp. 171–185, 1976.

[42] R. Barbi, D. Burihabwa, P. Felber, H. Mercier, and V. Schiavoni, "RECAST: Random entanglement for censorship-resistant archival storage," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Luxembourg City, Luxembourg, Jun. 2018.

[43] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed. Reading, MA, USA: Addison-Wesley, 1994.

Hugues Mercier received the B.Sc. degree in mathematics from Université Laval, the M.Sc. degree in computer science from the Université de Montréal, and the Ph.D. degree in electrical and computer engineering from the University of British Columbia, Canada, in 2008. From 2008 to 2011, he was a postdoctoral research fellow at the Harvard School of Engineering and Applied Sciences, and at McGill University. Currently, he is a research associate at the Université de Neuchâtel in Switzerland. His current interests are the applications of coding theory, information theory, combinatorics and algorithms to the study and design of communication and storage systems.

Maxime Augier received the M.Sc. degree in communication systems in 2008, and the Ph.D. degree in computer science in 2016, both from École Polytechnique Fédérale de Lausanne, Switzerland, He is currently working in industry, on topics related to security, programming languages and distributed systems.

Arjen K. Lenstra is Professor of Computer Science at École Polytechnique Fédérale de Lausanne. His research focuses on cryptography and computational number theory, especially in areas such as integer factorization. He was closely involved in the development of the number field sieve method for integer factorization as well as several other cryptologic results. He is the recipient of the Excellence in the Field of Mathematics RSA Conference 2008 Award and a Fellow of the International Association for Cryptologic Research (IACR).