Trustworthy Cloud Storage

THÈSE N^{O} 6976 (2016)

PRÉSENTÉE LE 7 AVRIL 2016

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

LABORATOIRE DE CRYPTOLOGIE ALGORITHMIQUE

PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Maxime AUGIER

acceptée sur proposition du jury:

Prof. M. A. Shokrollahi, président du jury
Prof. A. Lenstra, directeur de thèse
Dr H. Mercier, rapporteur
Prof. P. Junod, rapporteur
Prof. B. Ford, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2016

Fly, you fools!.
— Gandalf

To Marie-Christelle...

Acknowledgements

First and foremost, I would like to thank my advisor, Prof. Arjen Lenstra, for gifting me with the opportunity to work at LACAL, and for giving me the freedom to explore foreign topics, even though they were not the most closely related to the lab's core research activities.
I owe an eternal recognition to my coauthor, Dr. Hugues Mercier, for believing in my ideas and helping me mold them into proper science; and for giving me the courage to complete this thesis, which would have never come to fruition without his unending encouragement and support. Thanks to Prof. Amin Shokrollahi, Prof. Bryan Ford, and Prof. Pascal Junod for the valuable time they accepted to dedicate by participating in the jury committee, and for their feedback.
Thanks to Prof. Phil Janson, for his always present benevolence, and for the many years of fruitful collaboration on teaching activities. Teaching was by far the most personally rewarding endeavour I undertook as a PhD student.
Thanks to my former and present lab colleagues: Dag Arne Osvik, Thorsten Kleinjung, Joppe Bos, Anja Becker, Nicolas Gama, Andrea Miele, Alina Dudeanu, for their ever-present friendliness; to my office mates Paul Bottinelli, Marguerite Delcourt, and Youssef El-Houti; and to Mohammad Dashti and Yannis Klonatos, for their invaluable help as TAs. Thanks also to Thierry Hayoz, Guillaume Martres, Théophile Studer and Raphaël Frei, the master students whom I had the privilege to supervise, for investing their time into my project proposals. In general, thanks to all the wonderful other students I had the chance to teach at EPFL.
Greetings to all my hacker friends from IRC, always willing to discuss bleeding-edge topics, in no particular order: ark, Blaque, Delraich, eggman, Enila, ex0ns, Freshgale, fzn, gruik, IvyAlice, lixette, MaDG, mit, Notfound, OdyX, ploum, sebseb01, Serwyn, Snorky, supersnail, Tengu, Triskel, Trium, undert, varouschka, vicky, vomoho, and Wobu.
Things would not have sailed so smoothly without the constant help of the administrative staff of the school, and in particular of Monique Amhof, who was always there to help us out of inextricable situations.
Thanks also to my current employer, supervisor, and colleagues, for their flexibility and their indirect support in the completion of this thesis.
And last but not least, thanks to my wife, Marie-Christelle, who put up with the random work hours and math-induced insomnia; to my parents, Jean-Pierre and Tovy, my brother Alex, and all of my family, in particular to my uncle Shalom for sharing his love of mathematics; and to my two cats, Wifi and Ada, for cheering me up in times of need.

Lausanne, December 8th 2015 M. A.

Abstract

The Cloud trend is an attempt to leverage economics of scale in the domain of computing resources. Unfortunately, this often means losing control of the lower levels of a computer system, and exposing users to new threat vectors. These threats may be significant enough to forbid the use of clouds, and force giving up on their economical advantages.
Chapter 1 introduces some issues with current cloud storage systems, that should be fixed before a cloud storage system can be considered as safe as a self-managed system. Among these, we will focus on censorship resistance. We also explain the not immediately obvious way in which they relate to issues discussed in the last two chapters.
Chapter 2 formally defines censorship-resistance and describes the STEP-archive, an abstract model for a generic class of censorship-resistant storage systems. Within this model, we expose an asymmetry in hardness between attack algorithms (trying to perform censorship) and defense algorithms (trying to repair censored files). We discuss ideal choices for the many parameters and derive useful mathematical bounds when possible. We also simulate the behaviour of an ideal storage system to obtain experimental evidence of the effect of these parameter choices. We show that this model exhibits several counter-intuitive properties.
Chapter 3 deals with the issue of incorrect key generation. Cryptography being an essential component of our proposed secure storage system, we discuss common pitfalls in implementations of popular asymmetric cryptographic algorithms, and evidence of their presence in real-world implementations.
Chapter 4 discusses an operational aspect of storage systems, the choice of a block storage unit, and the consequences of lack thereof. In particular, it shows how the size of a ciphertext can act as a side channel and leak information about encrypted contents to an attacker, within the context of large media files distributed through public file sharing systems.

Key words: censorship, cloud, coding, cryptography, storage.

Résumé

Le Cloud est une tentative d'exploiter l'économie d'échelle dans le contexte des ressources de calcul. Malheureusement, cela signifie souvent renoncer au contrôle des couches inférieures du système informatique et exposer ses utilisateurs à de nouvelles opportunités d'attaques. Ces nouvelles menaces peuvent constituer des risques conséquents, assez pour renoncer aux avantages des services de cloud.

Le Chapitre 1 est une introduction aux limitations des systèmes de stockage en cloud actuels, lesquelles doivent impérativement être dépassées pour qu'un système de stockage en cloud puisse offrir un modèle de sécurité équivalent à un système auto-hébergé. Parmi celles-ci, nous nous intéresserons plus particulièrement à la résistance à la censure. Nous expliquerons aussi les liens non évidents entre ces limitations et les sujets abordés dans les deux derniers chapitres.

Le Chapitre 2 donne une définition formelle de la résistance à la censure et décrit l'archive STEP, un modèle abstrait représentant un système de stockage généraliste et résistant à la censure. Dans le cadre de ce modèle, nous montrons d'intéressantes asymétries entre la difficulté des algorithmes d'attaques (exécutant une opération de censure) et de défense (réparant les données perdues par la censure). Nous discutons du choix optimal des nombreux paramètres du modèle et établissons quelques bornes mathématiques. Nous étudions également son comportement par des simulations et validons expérimentalement certains choix de paramètres. Nous montrons certains aspects par lesquels ce modèle se comporte de manière contre-intuitive.

Le Chapitre 3 traite du problème de la génération incorrecte de clefs cryptographiques. La cryptographie étant la pierre angulaire de tout système de stockage sécurisé, nous y abordons les pièges courants dans l'implémentation des algorithmes asymétriques les plus populaires, et montrons des preuves de leur existence dans le monde réel.

Le Chapitre 4 traite d'un aspect opérationnel des systèmes de stockage, à savoir le choix d'une taille de bloc, et de ses conséquences. En particulier, nous déterminons si et comment la taille d'un message chiffré peut constituer un canal auxiliaire et révéler de l'information à un attaquant passif, dans le contexte de la distribution publique de grands fichiers multimédia.

Mots clefs : censure, cloud, codage, cryptographie, stockage

Thesis Bibliography

Chapter 2 is an extended version of the article:

Hugues Mercier, Maxime Augier, and Arjen K Lenstra. Step-archival: Storage integrity and anti-tampering using data entanglement. In *Information Theory (ISIT), 2015 IEEE International Symposium on*, pages 1590–1594. IEEE, 2015

Chapter 3 is taken from the article:

Arjen K Lenstra, James P Hughes, Maxime Augier, Joppe W Bos, Thorsten Kleinjung, and Christophe Wachter. Public keys. In *Advances in Cryptology–CRYPTO 2012*, pages 626–642. Springer Berlin Heidelberg, 2012

Contents

List of Figures

List of Tables

1 Current Issues with Cloud Storage

Cloud computing is a trend full of promises. Offering a free market for computing resources, we can greatly increase hardware utilization; we also save on power, cooling, and communication costs through the use of shared systems, concentrated into tight physical locations. Concentration allows saving on power and cooling (large-scale electrical transformers and cooling systems are more efficient), and on networking (much higher bandwidth can be achieved over short distances).

Yet, by doing so, we open Security Pandora's box, for information now flows and is processed by systems owned and controlled by third parties, possibly located in other countries with different legislation.

Ideally, a perfect cloud-based service should present the same threat model as a system owned and managed directly by the customer. This should be true even if the customer does not fully trust the cloud provider.

Secure remote general-purpose computation is a difficult problem. Key progress was made by [G+09], and numerous optimizations have been further developed, but they still impose several orders of magnitudes of computational overhead, and are thus impractical in all but the most extremely unbalanced scenarios (where the computing power of a cloud service dwarfs the power available to the end-user).

Instead of tackling this difficult general problem, this work is restricted to the much simpler scenario of trusted storage on third-party systems. The untrusted third-party system is not required to perform arbitrary computations, but just to store and retrieve opaque data.

1.1 Cloud Requirements and Trust Model

Well-understood cryptographic techniques, properly used on the client side, can already protect data confidentiality, integrity, and authenticity. However, there are other desirable properties relating to more exotic issues:

- Ensuring reliability from a third-party storage provider is harder than ensuring it on a privately-owned system. While in the latter case one only has to guard against accidental failures, in the former the storage provider may have an economic incentive to lie about its actual reliability. This issue can be addressed with proof-of-storage protocols. A trivial proof-of-storage protocol is to periodically perform a full retrieval of the contents; more advanced protocols, like [ZX12, JKJ07], can give a good probabilistic assurance that the contents are retrievable, at a much lower bandwidth cost.

- For content that is meant to be distributed to a wide audience, centralization creates opportunities of arbitrary censorship by the storage provider. Even if the provider is not itself willing to censor content, it may be coerced into it by a legal authority, or become the target of blackmail by other powerful attackers. These censoring threats may occur at unpredictable times and under great pressure. Defending against them is harder than in the previous case, where it is sufficient to ensure that the provider has, on average, an economic incentive to behave correctly. Chapter 2 investigates a possible direction for censorship-resistant storage systems.

- Beyond censorship, anonymity may be an important requirement. A cloud storage provider, if it is in position of knowing the identities of all the content publishers using its service, may become a prime target for various surveillance-oriented attackers. Usage of cloud storage resources implies usage of network resources. Systems like Tor [DMS04] partially help, but there are still side-channel threats to anonymity when large amounts of data move around.

- Anonymity of the content author is one thing when volunteers agree to provide free storage space. When storage space must be paid, however, one must be careful to ensure anonymity of the payment trail as well.

1.1.1 Comparison with peer-to-peer systems

Distributed peer-to-peer storage systems exhibit good robustness properties against an attacker trying to perform censorship, assuming the attacker cannot control the majority, or even a significant fraction, of the participating peers. This assumption may no longer hold when a majority of peers run or store data on cloud systems under control of a small number of large service providers. Therefore, it is not sufficient to simply run a peer-to-peer system on top of cloud resources, however tempting it may be. We wish to design a system with different guarantees, that can resist censorship without precluding the use of centralized cloud resources.

1.1.2 Mutability

Mutability refers to the way a storage system handles data getting modified over time. A mutable system offers an interface to store and read data objects, but also to modify all or part

of it at a given point in time. It means the system must also handle some form of locking or conflict resolution, should modification requests occur concurrently.

By contrast, an immutable system only supports creation of new data objects, optionally deletion, and reads. Modification can be emulated by creating a new, separate, version of the object, then dropping the old one. For many use cases, this may be much less efficient than a mutable interface; a small modification to a large object will wastefully recreate an almost identical copy of the object. However, an immutable model is also safer and friendlier: neither locking nor conflict resolution are required, it suits concurrent operation better, and it makes it easier to implement cryptographic integrity checks.

Chapters 2 and 4 both consider a system operating on top of cloud services offering an immutable storage interface.

1.1.3 Deduplication

Deduplication is the practice of recognizing redundant data blocks in a storage system, and save space by storing only a single copy. This may be done at different granularities: at the file level, with fixed data blocks, or with more complicated schemes like rolling hashes to split data into variable-sized slices.

For a large storage provider, deduplication may be an integral part of the business model: the data is stored once, but customers may be billed multiple times. This is especially worthwhile for large contents that get shared without any modifications, for instance multimedia files.

Deduplicating mutable storage is more expensive; a copy-on-write mechanism must be used to distinguish previously identical copies when one of them is getting modified. No such issue exists with an immutable interface.

- File-level deduplication is easy to implement in an immutable system. If every file is indexed by a cryptographically secure hash, it is natural to use this index both for retrieval and deduplication.
- Block-level deduplication operates on blocks of fixed size. It is typically much more expensive than file-level, if only because the number of objects to consider is much higher. It is however more interesting in some specific scenarios, typically with big files containing common substructures with predictable alignment. This is notably the case for filesystem images, used to distribute or store virtual machines, and optical media images (for instance rips of DVDs).
- Variable-block-level (sometimes inaccurately called *Byte-level*) uses techniques to slice a file into pieces of variable sizes, ideally with piece sizes following a geometrical distribution, in a way that makes the pieces locations depend only on the piece contents, and possibly the close surroundings of the piece. This makes it possible to share common

contents even if they appear at unaligned offsets in different files.

Rolling hashes

One early technique to find local split points is the Lyndon factorization, which splits a sequence of symbols (bytes, words, or bigger blocks) into a list of the largest possible sub-sequences, such as every sub-sequence has the property of being, of all its cyclic rotations, the one that comes first in lexicographic order. Such a factorization is unique and the split points can be computed in linear time.

Another technique is to decide on the split points with a rolling hash. Contrary to what the name suggests, a rolling hash is not a cryptographically secure hash, but rather a fast digest that can be efficiently computed from an old version when bytes are appended at one end of the data, and removed at the other.

This way, it is possible to efficiently compute the value of a fixed-size chunk of data within a file at every possible location, with low memory usage. Every location that exhibits a rolling hash of a special form will become a distinguished point over which the file can be sliced.

With this technique, if different files share a large part of identical data, even at unpredictable and unaligned offsets, the distinguished points are guaranteed to be identical everywhere but on the two borders of the identical parts, within two windows of the size of the rolling hash window. All the slices between the two furthest distinguished points will be considered identical and deduplicated.

1.2 The case of Tahoe

Tahoe is an example of a secure cloud filesystem with good security properties. It uses an immutable storage model for data, and optionally mutable metadata. Encryption is done client-side, and data is always authenticated. Data is stored redundantly across several nodes, using Reed-Solomon coding over stripes within a file. The stripes themselves have arbitrary length, and are encrypted with a stream-like CTR mode.

Tahoe provides neither a censorship-resistance mechanism, nor a proof-of-storage mechanism. Access control is done by capabilities, so in a basic setup there is no need for identity management. However, in such a basic setup, storage space is a shared resource between participants, and freeloaders can use more space than what they contribute.

1.2.1 Extending to Censorship Resistance

Chapter 2 describes the STEP-archive, an abstract model for a generic censorship-resistant storage system, that extends and covers the behaviour of some early censorship-resistant systems like Tangler [Wal01] and Dagster [SW01]. It provides both censorship resistance and

redundancy as two integrated properties.

Inspiration for these early systems came from an interesting practical realization: in many parts of the world, some data may be illegal to possess. To the layman, it is not obvious whether a storage provider should be liable for allowing customers to store illegal data. However, it makes no sense to criminalize possession of *random* data.

Suppose a customer encrypts illegal data using a one-time pad, and stores both the key and the ciphertext on two different storage providers. If the key has correctly been chosen truly randomly, it is impossible[1] to distinguish the ciphertext from the key. Thus, even if one considers possession of encrypted data to be reprehensible, it is impossible to put the blame on one or the other provider. Furthermore, the key may also have been already present in the system for other reasons, instead of having been generated for the occasion. Unless one can prove that one side of the storage (the key) predated the other (the ciphertext), then one must consider that at least one provider may not even have been willing to cooperate with the law-breaking customer.

This property is not unique to one-time pad encryption. It also applies, for instance, to the non-systematic error correction codes we use in Chapter 2.

1.2.2 Key proliferation by Mutability

For mutable metadata, updates are authenticated by binding an RSA public key to the mutable metadata identifier, and signing all new (immutable) versions of the data with the corresponding private key. This leads to a proliferation of RSA keypairs. Chapter 3 discusses the possible pitfalls of incorrect key generation, and in particular pitfalls exacerbated when keypairs are being massively generated.

1.2.3 Storage incentives and Anonymity

An obvious way to defend against freeloaders is to account for space usage by all the participants in a Tahoe grid. Such accounting defeats anonymity.

We have considered an anonymous payment protocol based on Chaum's Blind Signatures. Blind Signatures are introduced in [Cha82] as a building block for an anonymous, offline, electronic cash payment system.

Offline electronic payment systems need a way to prevent double spending. In [Cha82], double spending is detected *a posteriori* by a complicated probabilistic protocol that lifts the anonymity of cheaters with high probability. However, in the case of immutable storage systems, a very interesting improvement can be made: since an immutable block of data can already be identified by a secure hash, we can use this hash instead of a random nonce when

[1] In the information-theoretic sense.

generating the blinded cash coin. This makes double spending irrelevant: a user may try to cheat by paying several times with the same coin to store the same data block, but if the storage system performs deduplication, it allocates resources only once, and no value is lost[2].

Unfortunately, in practice our naive attempts at such a protocol suffered from side-channel attacks. Because the blinded coin cannot be generated before the data hash is known, it is not possible to pay for storage in advance. Thus there would be strong time correlations between the space payment transactions and the data uploads, defeating the purpose of anonymous payments. However, because the payment messages are small compared to the actual data, and because a storage provider could certainly tolerate a non-realtime operation of the payment protocol, alternatives like [CGF10] may work better than Tor in this scenario.

Still, such a payment protocol would cause another proliferation of RSA keypairs, and another reason to look at the issues presented in Chapter 3

1.2.4 Variable block sizes

Tahoe uses variable-length stripes to store data. For efficiency, the encrypted file is split into fixed-size segments, and the segments are then striped across several *primary shares*, of identical length (up to the segment size), with the sum of the primary share sizes being roughly equal to the size of the encrypted file. An error-correcting code is then applied to compute a number of *secondary shares*, having the same size as the primary shares. Authentication and integrity checking range over the entire share; it is not possible to address a segment individually (although segments may be integrity-checked individually through a Merkle tree).

For the purpose of the STEP model considered in Chapter 2, it is easier to consider only storage of fixed-size blocks, and consider that the underlying storage system addresses blocks, not entire stripes.

While it may seem trivial to impose a fixed block size, by adding a block layer on top of the system, such a change is not benign in terms of security. Chapter 4 deals with the implications, including a possible subsequent leak of confidentiality.

1.3 Conclusion

This concludes our overview of current issues. We hope that the STEP construction constitutes a sound basis for bringing censorship resistance properties to cloud storage. Success in implementing these properties could lift some of the current barriers to adoption of cloud platforms, and allow users to benefit from power and bandwidth savings even for security-sensitive applications.

[2] This only works if the storage provider gives up on the practice of gouging users by making them pay an individual price for a shared resource.

2 STeP-archival: A Storage Model for Censorship-Resistance

2.1 Introduction

The way we store and archive data is being transformed by the emergence of information and communication technologies. These new tools open worlds of new opportunities but at the same they time pose significant challenges. 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 the 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 must transit outside the legal jurisdiction of its rightful owner.

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

A second challenge of digital storage and permanent archiving is tamper resistance. This is closely related to protection against censorship. Research and scientific data as well as medical, legal and financial records can include very sensitive information that can be viewed

as threatening or compromising by potential censors. A good archival system must thus make it very difficult for a powerful censor to irrecoverably destroy or tamper with archived data, especially in an undetectable way. This is a different issue from the traditional definition of data integrity and authenticity, for which there already exist plenty of solutions from the client perspective using client-side cryptography. For compliance and legal reasons, such sensitive data may be stored using a "write-once, read-many" (WORM) technology. WORM storage is a niche market of secure data storage solutions that has been historically fulfilled using hardware approaches. Physical implementations offer more constrained data access than logical approaches and are more dependent on hardware robustness against failures and destruction. The implementation of software approaches for anti-tampering is an active topic of investigation and no satisfying solution is available in practice.

Censorship effectiveness is highly dependent on the communication technologies used to spread the censored information. In our modern world of electronic communications, information flows in such volume that human-based censorship has become impossible. Information processing systems can, however, be used for automated censorship. The emergence of ubiquitous, interoperable communication networks makes it easier to implement global automated censorship.

The extreme impracticality of censoring mouth-to-ear communication ultimately makes censorship a lost cause: Determination to transmit a message is the only requirement for eventually finding a way. However, large-scale automated censorship can still have damaging consequences. Fear of censorship or surveillance may hinder the adoption of otherwise promising communication technologies.

This issue is prevalent in the context of cloud storage, where a storage service provider could be pressured into removing specific contents, against the will of their customers. Beyond legal pressure, employees could also be bribed or coerced into damaging customer data. Fear of data losses may drive customers away from cloud storage and its cost-saving benefits.

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

2.1.1 State of the art

Data integrity, protection against tampering and anti-censorship have been studied in various forms and for a large number of settings and applications. Work on these topics include the Eternity Service [And96], Publius [WRC00], Freenet [CSWH01], Free Haven [DFM01], Dagster [SW01], Tangler [Wal01], SiRiUS [jGSMB03], Tahoe [WOW08], Clouds [BHL+08] and POTSHARDS [SGMV09].

Randomized encryption can be used to prevent a malicious storage system from extracting information about its users by observing with whom they share files. It can also be used to

prevent the system from preemptively censor documents corresponding to known content. This has been implemented with success in practice, notably by the Tahoe [WOW08] filesystem under the name of Convergent Encryption. Encryption keys are semi-deterministically derived from the hash of a cleartext block so that efficient deduplication can still be performed on encrypted blocks to reduce storage space. However, a third party could publish the encryption key for a particular block, and prove that some block decrypts to censorable content, prompting an authority to individually censor particular blocks.

Data integrity and resistance against tampering are not easy to define, especially when considering how to provide these features in a practical way. For instance, in [WRC00], the authors informally write "Our system should make it extremely difficult for a third party to make changes to or force the deletion of published materials". This was also observed in [PRW05], which provided a more formal definition of censorship resistance in the context of selective filtering. A definition of data integrity was made in [AFYZ07], but in such a strong way that it cannot be achieved in practical systems.

Other interesting related work includes plausibly-deniable search [17912] and proofs of storage and retrievability [ZX12, JKJ07]. In [JKJ07] the authors describe an efficient proof of retrievability mechanism that allows a client to verify the existence of a piece of data in a storage system. The authors correctly note, however, that such a mechanism cannot guarantee that the system will agree to disclose the actual data when prompted to do so.

In [AFYZ07], the authors studied data integrity and developed a theory of data entanglement. One of their contributions is the introduction of all-or-nothing integrity: intuitively, either all the documents are recoverable with high probability, or no document is. They show that all-or-nothing integrity is possible with some restrictions on the power of the attacker. [ADDV15] extended the work by providing a stronger definition of all-or-nothing integrity and a simulation-based security analysis. The protocols provided in both articles [AFYZ07, ADDV15] remain far from real-life implementations: they require to read the entire data store to retrieve a document, and require to process the entire data store to add a new document, which is not scalable. Furthermore, no document is recoverable if the storage provider corrupts or fails to maintain a small part of the data.

Providing anti-censorship using data entanglement was first proposed by Dagster [SW01] and Tangler [Wal01], which both can be seen as special cases of the approach we propose in this chapter. In Dagster, documents and blocks have the same size. To add a new document in the system, c blocks already stored are chosen at random, and a new block consisting of the exclusive-or of the new document with the c blocks is stored. A censor wanting to delete a document can erase one of its $c+1$ blocks, and on average over all documents this will destroy on average $O(c)$ other documents, the older documents being more protected than newer ones. In Tangler, two old blocks chosen randomly and a new document to be archived are used to generate two new blocks using $(3,4)$ Shamir secret sharing [Sha79]. The two new blocks are then stored. The original document can be recovered with any three of the

four blocks using Lagrange interpolation. In [AFYZ07], it was shown that erasing two blocks from a random Tangler document erases on average $O(\frac{\log n}{n})$ 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.

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

2.1.2 Our contributions

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

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 collateral damage is **NP**-hard. Finding an irrecoverable attack whose collateral damage is less than optimal but within a reasonable ratio from the optimal solution is also **NP**-hard.

System robustness

In [AFYZ07, ADDV15], it is 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 the perverse effect that a malicious attacker, having compromised an honest provider, can deny the 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: when a document

has been archived long enough, it can only be lost by destroying a large number of other documents. However, 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.

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 documents archived after it.

Practical considerations

We emphasize that our objective is to achieve both data integrity and censorship resistance in a way implementable in practical systems. Thus, we use practical constraints that keep an actual implementation realistic (for instance avoiding reads and writes in $\omega(1)$ of the total system size) while being simple enough to allow analysis. All our underlying assumptions and design choices are implementable using state-of-the-art coding and storage techniques.

2.1.3 Outline

The rest of this chapter is organized as follows. In section 2.2, we present generic assumptions and constraints. The storage architecture is formally described in Section 2.3, and the recovery algorithm in Section 2.4. We analyze the optimal attack in Section 2.5, and describe suboptimal attacks in Section 2.6. Two entanglement strategies, proximity entanglement, and uniformly random entanglement, are respectively studied in Sections 2.7 and 2.8. In Section 2.9, we discuss extensions and general security considerations. Finally, we conclude the paper in Section 2.10.

2.2 Model assumptions and architecture constraints

In this section we describe the main assumptions and constraints used throughout this chapter. They are simple and realistic for a first contribution on this topic. However, it should be clear that they can be relaxed and extended in various ways, as discussed in Section 2.9.

2.2.1 Censorship resistance

Our goal is to offer a storage system providing some level of *censorship resistance* (CR) implementable both with centralized and decentralized architectures, in the sense that security guarantees still hold when most hardware resources are under the control of a single entity.

A censorship resistant system makes it difficult, even for the entity in control of the system, to silently and selectively refuse to answer particular requests without denying service for other unrelated requests. We precisely define three levels of censorship resistance.

- ***Perfect CR***: Either all the read requests can be fulfilled, or none can. This is similar to all-or-nothing integrity as defined in [AFYZ07].
- ***Strong CR***: If the system is unable to fulfill a read request, then a constant fraction of possible read requests cannot be fulfilled (*collateral damage*).
- ***Weak (resource driven) CR***: The system must spend an amount of hardware resources proportional to the size of the system to censor a read request.

In ultimate recourse, the service provider can always be forced to shut down entirely. Our definition of *Perfect CR* follows from this observation: in a perfect system, there is no better way to censor data. The difference between *Strong CR* and *Weak CR* is that the former guarantees that a censorship attempt will have an impact on the system clients and cannot go unnoticed, whereas with the latter case an attacker could coerce the system to perform censorship, coercion invisible to other users.

We emphasize that we tackle data integrity, tamper resistance and censorship resistance at the same time, and thus use all terms interchangeably. For instance, with Perfect CR, every document in the archive is as reliably archived as every other document, thus the integrity of every document is the same. We formally discuss the equivalence of data integrity, tamper resistance and censorship resistance in Section 2.3.

2.2.2 Storage interface

We will achieve data integrity and protection against tampering with a coding scheme for which unrelated pieces of data will become mutually dependent. Since we are looking for efficient and practical implementations, our first constraint is that we only consider solutions that can archive a new document using only a small constant amount of data already archived.

Our second constraint is the use of an immutable data store as the underlying storage structure. In this setting, updates effectively become new documents. This is a reasonable assumption for a large class of applications. The two constraints work in tandem in the sense that any system that allows to rewrite existing data easily has weak integrity and little protection against tampering.

Our third constraint is that the metadata is readable by a malicious attacker, who knows which data blocks belong to each document. The attacker also knows the creation date of each data block and can therefore order them chronologically. Our fourth constraint is that the system operates on fixed-size data blocks.

Finally, our fifth constraint is that the communication protocol provides self-integrity, that is, when fetching the key of a stored block B, 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 key computed as a cryptographically secure hash function of B, and have the client recompute the hash from the data and check the matching key after every read operation. A malicious server must be able to break the hash function in order to break this self-integrity. A related assumption is that the attacker can tamper with data blocks, but not with metadata. Thus, after an attack or failure, the system knows from the metadata which data blocks have been corrupted or tampered with, i.e., the errors are erasures. We examine the consequences of relaxing these assumptions in Sections 2.9 and 2.10 .

2.2.3 Snapshotting

If the state of the system can be cheaply snapshot, then tampering can be implemented by reverting the system to an earlier state not containing the tampered data. Thus, with immutable data structures, the best system will at most guarantee that data can only be tampered by destroying all data stored *after* it. Perfect censorship-resistance is unattainable in this context. Although this can be seen as an undesirable property, we argue that we cannot achieve perfect CR without violating our first constraint. Moreover, 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 anti-tampering and integrity quickly on a time scale.

2.3 Entanglement architecture using erasures codes

Definition 1. *A (s, t, e, p)-archive is a storage system where each archived document consists of a code word 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 [LC04] is used to create $p \geq s$ parity blocks which are then archived on the system. The process is drawn in figure 2.1.

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

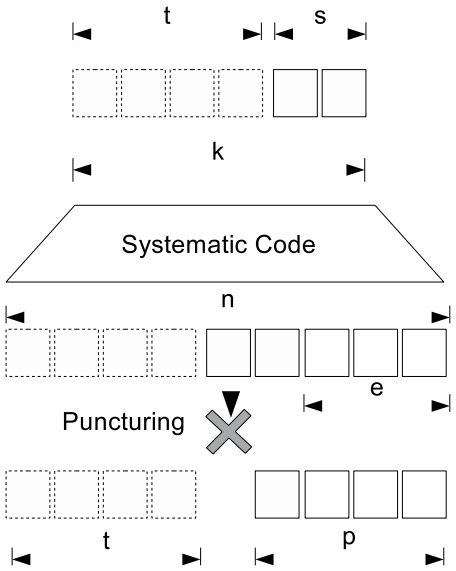
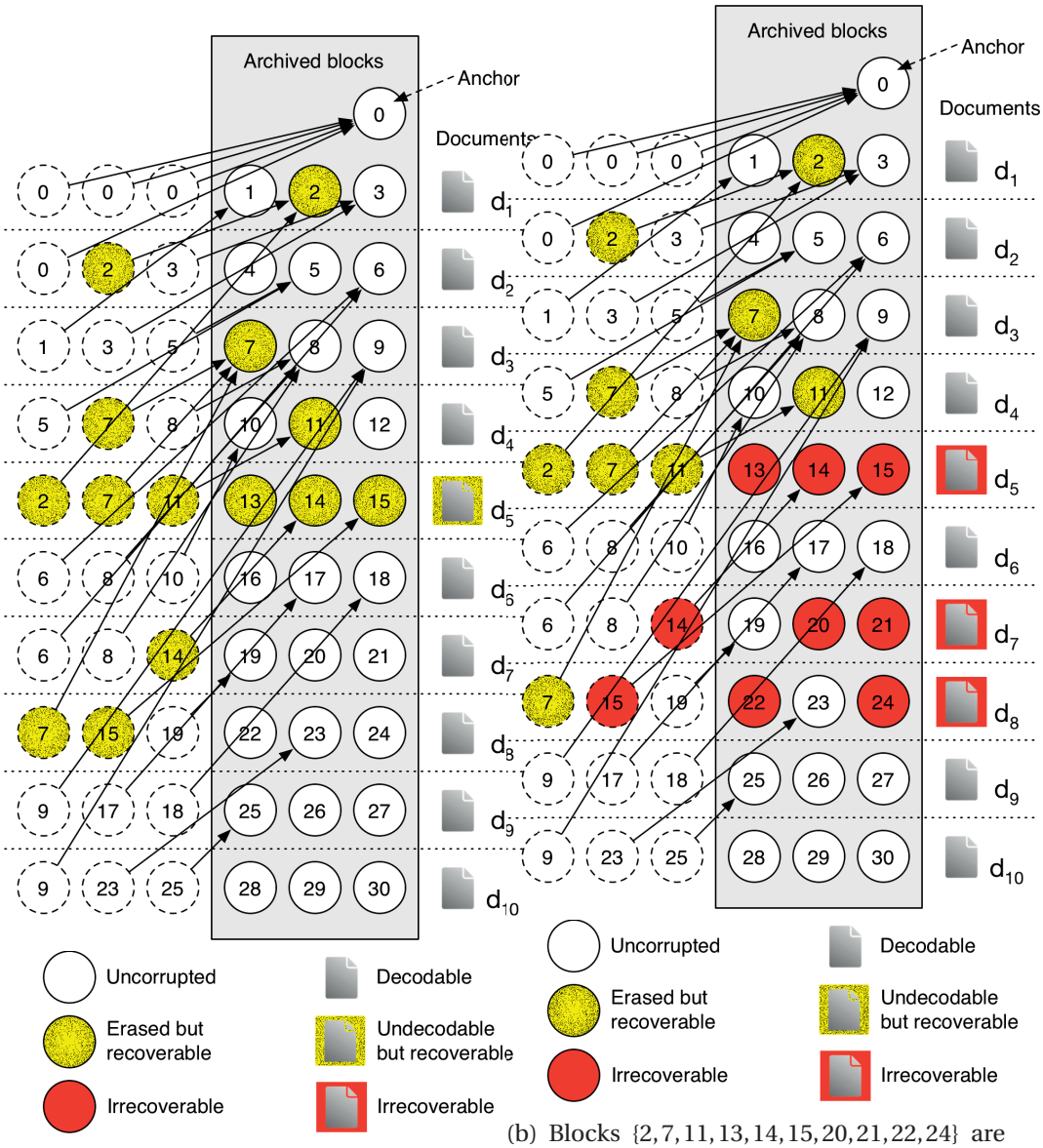


Figure 2.1 – Encoding of a single document in STEP-archival

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 increase storage overhead or error-correcting capability, but does increase coding and decoding complexity.

An attacker can censor a document d_k by erasing more than e 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 2.2. 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 2.2a, an attacker wants to censor document d_5 by erasing its blocks $\{2,7,11,13,14,15\}$ from the archive. However, although d_5 cannot be recovered directly, all the blocks are recoverable: Block 2 can be recovered by decoding d_1 or d_2, Block 7 can be recovered by decoding d_3, d_4 or d_8, Block 11 can be recovered by decoding d_4, Block 14 can be recovered by decoding d_7, Block 15 can be recovered by decoding d_8, and in the last step Block 13 can be recovered by decoding d_5. Having been unable to erase d_5, the attacker continues his attack more cleverly and further erases Blocks 20, 21, 22 and 24, as illustrated in Figure 2.2b. 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.

[1] 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.



(a) Blocks $\{2,7,11,13,14,15\}$ are erased.

(b) Blocks $\{2,7,11,13,14,15,20,21,22,24\}$ are erased.

Figure 2.2 – (1,3,2,3)-archive. 4 out of 6 blocks are required to recover a document.

Definition 2. *A set of documents form an* integrity set *I if and only 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 $\mathscr{A}$ 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 $a \in \mathscr{A}$, $\mathscr{I}_{min}(d_k) \triangleq \min_{1\leq j\leq k}(I_{min}(d_j))$ for the size of the smallest of the integrity sets of the first k documents for a fixed archive $a \in \mathscr{A}$, and ${}_{max}\mathscr{I}_{min}(d_k) \triangleq \max_{a\in\mathscr{A}}(\mathscr{I}_{min}(d_k))$ for the largest $\mathscr{I}_{min}(d_k)$ over all possible archives $a \in \mathscr{A}$. We write $\mathscr{I}_{min}$ and ${}_{max}\mathscr{I}_{min}$ when $K \gg k$.*

Note that $\mathscr{I}_{\min}(d_k)$, ${}_{\max}\mathscr{I}_{\min}(d_k)$, $\mathscr{I}_{\min}$ and ${}_{\max}\mathscr{I}_{\min}$ are nondecreasing 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 $\mathscr{I}_{\min}$ 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 $\mathscr{A}$ 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 smallest integrity window of document d_k for a fixed archive $a \in \mathscr{A}$, $\mathscr{W}_{min}(d_k) \triangleq \min_{1\leq j\leq k}(W_{min}(d_j))$ for the size of the smallest of the integrity windows of the first k documents for a fixed archive $a \in \mathscr{A}$, and ${}_{max}\mathscr{W}_{min}(d_k) \triangleq \max_{a\in\mathscr{A}}(\mathscr{W}_{min}(d_k))$ for the largest $\mathscr{W}_{min}(d_k)$ over all possible archives $a \in \mathscr{A}$. We write $\mathscr{W}_{min}$ and ${}_{max}\mathscr{W}_{min}$ when $K \gg k$.*

[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 $\mathscr{A}$ is which tangled blocks are selected for each document.

[4] See Footnote 3.

From Figure 2.2b, 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 $\mathscr{I}_{\min}(d_5) = 3$ and $\mathscr{W}_{\min}(d_5) = 4$. $\mathscr{I}_{\min}(d_6) = 2$ since d_6 can be destroyed by deleting the blocks $\{16, 17, 18, 26\}$ in documents d_6 and d_9.

The challenging part of our approach is thus to choose the pointers to entangled blocks in a practicable way that provides good anti-tampering and data integrity. With a non-constant number of pointers per document, maximum forward data integrity and anti-tampering is possible: on an archive with $s = 1$ and $e \geq 1$, 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. Of course, the number of pointers to tangled blocks in this example grows linearly with the number of documents, which makes encoding and decoding too complex to be of any practical value. Since we target practically implementable data integrity and anti-tampering, we thus focus on archives with t constant and small.

2.4 Reconstruction algorithm

One of the interesting aspects of our approach is its asymmetry: while impossible for an attacker to find the optimal strategy to irrecoverably tamper a target document d_k in polynomial time (unless **P** = **NP**), repairing the system if the damage done is recoverable is easy and doable in linear time, as shown in Algorithm 1. 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 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 2.2a, the reconstruction algorithm can recover all the erased blocks, whereas in Figure 2.2b it can recover the yellow blocks {2,7,11} but is incapable to recover the red blocks

Algorithm 1: Reconstruction algorithm

input : $e \leftarrow$ erasure decoding capability of the code
$C \leftarrow$ set of corrupted documents in the archive

```
while C ≠ ∅ do
    pick an element d ∈ C
    decode d and recover its set of erased blocks B
    forall the archived documents d' with at least one block in B do
        if d' is corrupted and has at most e erased blocks then
            C ← C ∪ {d'}
    C ← C \ {d}
```

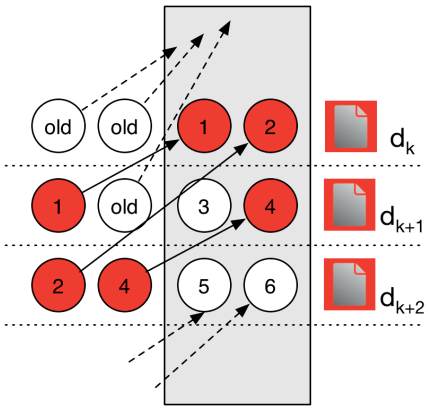


Figure 2.3 – (1,2,1,2)-archive with dependency cycle.

{13,14,15,20,21,22,24}. The algorithm is highly parallelizable: in Figure 2.2a, 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 2.3. 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. Furthermore, a single erased block that is not constrained by other documents is sufficient to ensure multiple solutions. In Figure 2.3, the attacker could also have erased Block 3, which breaks the dependency cycle by adding a degree of freedom.

2.5 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 an 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 an 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 $\mathscr{I}_{\min}(d_k)$.

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 Erdős et al. [EFRS90] and Bollobás and Brightwell [BB89]. More recently, [APP+12] 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 [PSS13], where the authors showed that for $d \geq 3$ and $\epsilon > 0$ there is no polynomial-time algorithm with approximation ratio $2^{\mathscr{O}(\log^{1-\epsilon} n)}$ unless $\mathbf{NP} \subseteq \mathbf{DTIME}\left(2^{\mathscr{O}(\log^{1-\epsilon} n)}\right)$, even with graphs with degree d or $d+1$. The authors also presented a polynomial-time randomized approximation algorithm with ratio $\mathscr{O}\left(\frac{n}{\log n}\right)$, and a brute-force polynomial-time deterministic approximation algorithm of ratio $\mathscr{O}\left(\frac{n \log n}{\log\log n}\right)$ for low-degree graphs. The parametrized complexity of the d-girth problem was studied in [ASS12], where the authors proved that the problem is $\mathbf{W}[\mathbf{1}]$-hard[5] for general graphs, but fixed-parameter tractable when graphs have bounded local tree width. From the discussion in [PSS13], 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*. However, 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.

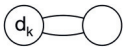
In this section, we consider a target document d_k archived long enough, i.e., with $K \gg k$ documents archived after it. We mention that for general hypergraphs, contrary to graphs, the 2-girth problem seems difficult as well, however we show that under certain conditions, there exists a polynomial-time 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 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 $2^{\mathscr{O}(\log^{1-\epsilon} n)}$ unless $\mathbf{NP} \subseteq \mathbf{DTIME}\left(2^{\mathscr{O}(\log^{1-\epsilon} n)}\right)$.

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

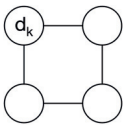
Proof. Let $G = (V, E)$ be a multigraph with loops. We consider the size g_2 of the minimum subgraph of minimum degree at least two that includes a target vertex d_k. We can find the 2-girth over the multigraph by repeating the algorithm for all vertices. Furthermore, although we omit the details, the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

[5] Consult [FG06] for a formal definition.

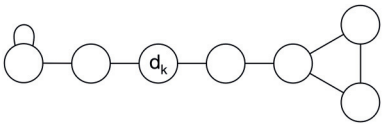
(a) 2 loops at d_k.



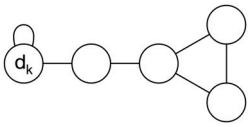
(b) Parallel edges from d_k.



(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.

Figure 2.4 – Types of minimum subgraphs of minimum degree at least 2 in a multigraph with loops.

The five types of minimum subgraphs of minimum degree at least 2 in a multigraph with loops are shown in Figure 2.4. If the multigraph has two loops at d_k (Fig. 2.4a), then $g_2 = 1$. If the multigraph has less than two loops at d_k but parallel edges from d_k (Fig. 2.4b), 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. 2.4c), 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. 2.4d), 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. 2.4e).

We now present Algorithm 2, 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

Algorithm 2: MSMD_2 with repeated vertices (Types 2.4d and 2.4e)

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: $\min(g_2^d, g_2^e)$

forall the $v_i \in V_i(d_k)$ **do**
  forall the $v \in V \setminus \{d_k \cup v_i\}$ **do**
    $p(v_i, v) \leftarrow$ length of shortest path from v_i to v
  $p(v_i, v_i) \leftarrow 0$
forall the $v \in V \setminus \{d_k\}$ **do**
  $c(v) \leftarrow$ length of shortest cycle including v
  `// The shortest cycle can be a loop or parallel edges if present`
if *there is no loop at* d_k **then**
  return $\min_{\substack{v_{i_1}, v_{i_2} \in V_i \\ v_{i_1} \neq v_{i_2} \\ v_1, v_2 \in V \setminus \{d_k\}}} (p(v_{i_1}, v_1) + c(v_1) + p(v_{i_2}, v_2) + c(v_2) + 1)$
else
  return $\min_{\substack{v_{i_1} \in V_i \\ v_1 \in V \setminus \{d_k\}}} (p(v_{i_1}, v_1) + c(v_1) + 1)$

from each vertex incident to d_k that ends with a 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 2.4e), and the sum of the two smallest such paths if d_k is in the middle of the path (Type 2.4d). For Type 2.4d, 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 2.4c). We conclude the proof by pointing out that all the steps of Algorithm 2 are executed in polynomial time. □

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. □

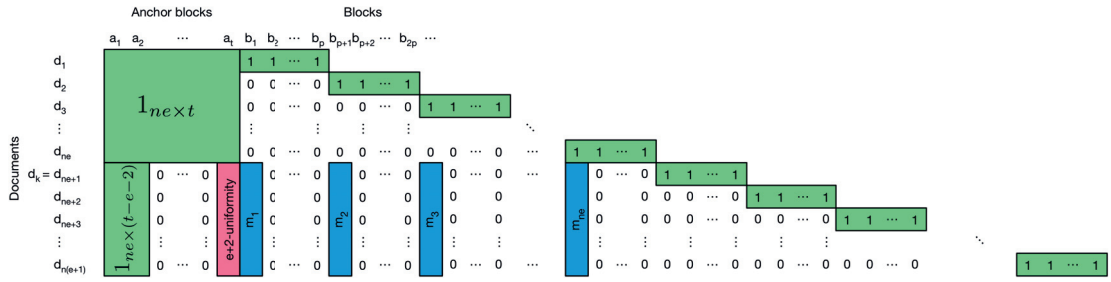


Figure 2.5 – Polynomial reduction from $I_{\min}(d_k)$ to the $e+1$-girth problem: incidence matrix for the top of the archive.

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. 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 $e \geq 2$ [APP+12, PSS13]. 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 2.5 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 2.5). 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.

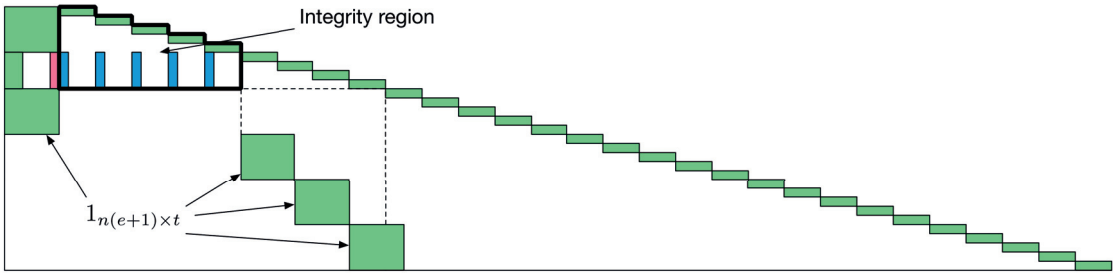


Figure 2.6 – Polynomial reduction from $I_{\min}(d_k)$ to the $e+1$-girth problem: incidence matrix for the entire archive.

Figure 2.6 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

$$n(e+1)\cdot\left(\left\lceil\frac{np}{t}\right\rceil+2\right)\in\mathcal{O}(n^2). \tag{2.1}$$

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 2.6. 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 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,

$$\begin{aligned} I_{\min}(d_k) &\leq \delta + \frac{\delta\cdot(e+2)}{2} \\ &\leq \delta\left(1+\frac{e+2}{2}\right). \end{aligned} \tag{2.2}$$

It is not possible for an integrity set of d_k to contain less 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

$$\begin{aligned} I_{\min}(d_k) &\geq \delta + \frac{\delta \cdot (e+1)}{2} \\ &= \delta\left(1 + \frac{e+1}{2}\right). \end{aligned} \tag{2.3}$$

Putting (2.2) and (2.3) together and solving for δ, we obtain

$$\frac{I_{\min}(d_k)}{1+\frac{e+2}{2}} \leq \delta \leq \frac{I_{\min}(d_k)}{1+\frac{e+1}{2}}. \tag{2.4}$$

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 $\mathbf{P} \neq \mathbf{NP}$. 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 [APP+12, PSS13] 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 $I_{\min}(d_k) = 1$. □

We mention to conclude this section that verifying that a set of documents is an integrity set is in **P** from the reconstruction algorithm, hence finding $I_{\min}(d_k)$ is **NP**-complete.

2.6 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 approximative 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. Thus, the tree is finite (and is, in fact, a lattice rather than a tree, 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.

2.6.1 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[6] number of blocks required to make the processed document impossible to directly decode. 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.

Minimum Attack

The minimum attack, described in Function 4, minimizes the set C of corrupted target documents, by always prefering blocks that are referenced by the least amount of documents not already in the target set.

Leaping Attack

The leaping attack, described in Function 5, is based on the intuition that it is easier to attack recent documents than older ones. We show in Section 2.8 that the leaping attack is especially suitable to damage a system built using an uniform random selection of pointers. The score of each block is 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

[6] as dictated by the code parameters, and the number of blocks already erased in earlier steps

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 ← {k}                                              // Corrupted but decodable targets
2  C ← {k}                                              // All corrupted targets
3  B ← ∅                                                // Erased blocks
4  while R ≠ ∅ do
5      r ← min(R)
6      while d_r has less than e + 1 erased blocks do
7          score ← ∅
8          forall the blocks b not erased in d_r do
9              score[b] ← heuristic(C, B, b)
                                                        // Available heuristics:
                                                      // MinimumAttack, LeapingAttack,
                                                     // CreepingAttack, TailoredAttack
10         b ← arg min_b score[b]
11         B ← B ∪ {b}
12         R ← R ∪ (documents(b) \ C)
13         C ← C ∪ documents(b)
14     R ← R\{r}
15 return B
```

Function 4: MinimumAttack(C, B, b)

```
1  return |C ∪ documents(b)|
```

referencing a block are already in C, the block score is the minimum of an empty set of integers, which we take as 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 2.3, the amount of parity blocks p is strictly greater than the number of correctible errors e, unless in the degenerate case $s = 0$ where the system would not be able to store any new information.

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.

Function 5: LeapingAttack(C, B, b)

```
1  N ← documents(b) \ C
2  if N = ∅ then
3      return −∞ ;                                      // free block
4  else
5      Return 0 − min(N)
```

Creeping Attack

The creeping attack, described in Function 6, intuitively tries to keep the set of corrupted documents C as compact as possible in time. We will see in Section 2.7 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.

Function 6: CreepingAttack(C, B, b)

```
1 C' ← C ∪ documents(b)
2 return max(C') − min(C')
```

Tailored Attack

Although we study pointers selected uniformly at random in Section 2.8, here we briefly 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 uniformly random entanglement, is described in Function 7 and further discussed in Section 2.8.

Function 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 ← 0
2 B' = B ∪ b
3 C' = C ∪ documents(b)
4 forall the c ∈ C' do
5     yettoerase = max(0, e + 1 − |B' ∩ blocks(c)|)
6     cost ← cost + 1 + yettoerase · ln(1 + (K − c)/c)^t
7 return cost
```

2.6.2 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

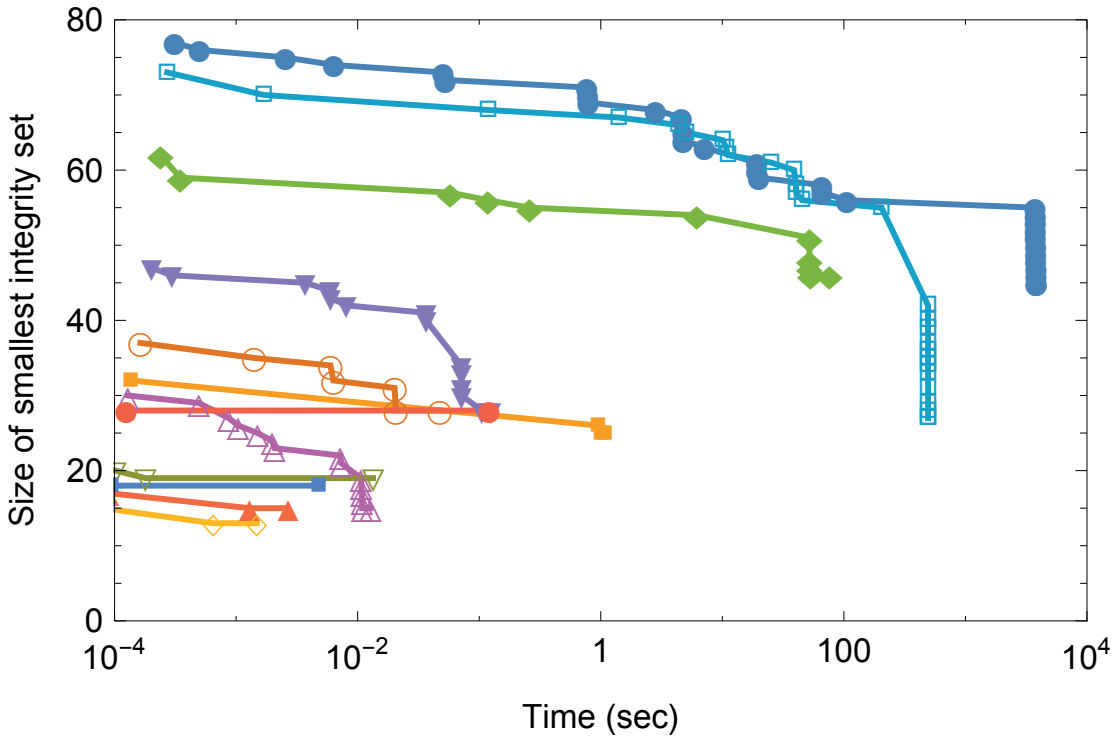


Figure 2.7 – Trace of the depth-first search leaping attacks for twelve $(1,3,2,3)-$archives with 10^4 documents, target document d_{10^3} and pointers chosen uniformly at random. The algorithm traversed the entire tree in $\approx 10^4$ seconds in the worst case.

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 2.7 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[7] is not proportional to the time spent attacking. Instead, the solution improves in unpredictible 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.

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

2.6.3 Bounded breadth-first search

The inefficiency of depth-first search motivates the investigation of bounded breadth-first search algorithm. For large systems, it is impossible 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.

2.6.4 Performance evaluation

In order to measure the effectiveness of our attacks and reconstruction algorithm, we simulated a storage system. The simulation keeps track of the dependencies between hypothetical blocks and documents, and the state (healthy or erased) of the data blocks. Recovery is performed according to Algorithm 1 by assuming an MDS code of given parameters; no actual data storage or decoding is performed.

2.7 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 documents, with an efficient attack to do so, concentrated over 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*

$$d_k \triangleq (t_k^1, t_k^2, \dots t_k^t, b_k^1, b_k^2, \dots, b_k^p). \tag{2.5}$$

Consider a (s,t,e,p)-archive with a sliding window of size w. In other words, 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 ${}_{\max}\mathscr{I}_{\min} = 2$ and ${}_{\max}\mathscr{W}_{\min} = w+1$. This can be achieved, for instance, 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 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 increase 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,*

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

Proof. We first prove the upper bound. If there are less 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 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, ${}_{\max}\mathscr{I}_{\min} \leq {}_{\max}\mathscr{W}_{\min} \leq 2w$

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

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

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,

$$W_{\min}(d_k) \geq \min(2w-p+1, 2w-p+2) = 2w-p+1. \tag{2.6}$$

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

$$_{\max}\mathscr{W}_{\min} \geq \mathscr{W}_{\min}(d_k) \geq 2w-p+1. \tag{2.7}$$

□

Surprisingly, having more than p entangled blocks per document does not appear to provide more integrity, and even becomes harmful as t increases, 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,*

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

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, $e+1$ of its p parity blocks must be erased. Since every block in d_k in used 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 make this attack more difficult: if the pointers are bounded to w documents in the past, then the anti-tampering protection is also bounded.

2.7.1 Random entanglement within a sliding window

With a small sliding window, we can explore the search tree exhaustively. To illustrate the previous results, Figure 2.8 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 box-and-whisker[8] plot for 1000 simulations. Figure 2.8a shows how the integrity sets vary, whereas Figure 2.8b focuses on the integrity windows. The green squares in Figure 2.8a represent the best regular entanglement within the window (see Section 2.7.2). 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.

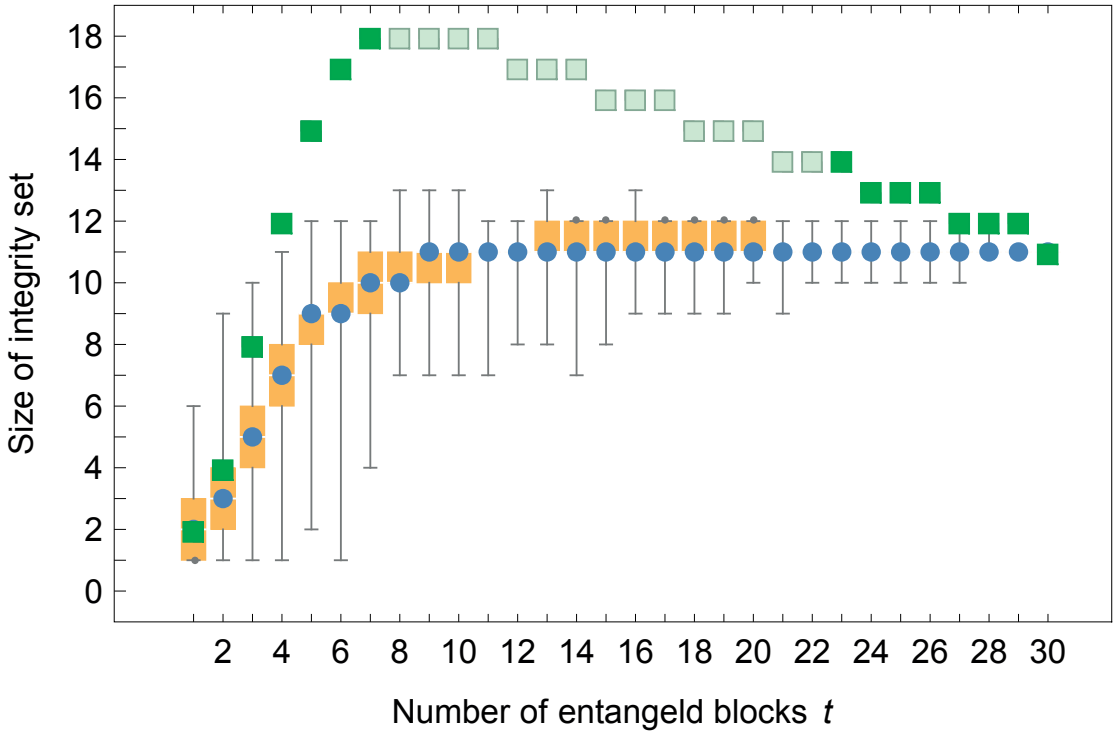
Figure 2.9 shows the result of $\min_{990 \le k \le 1010} I_{\min}(d_k)$ with the same simulation parameters. The attack sequentially targets d_i for $990 \le i \le 1010$, and shows the size of the smallest integrity set among the targeted documents. Compared to Figure 2.8, we can see that with few pointers, 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 2.8 and 2.9 have the same behavior.

Figures 2.10 and 2.11 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.

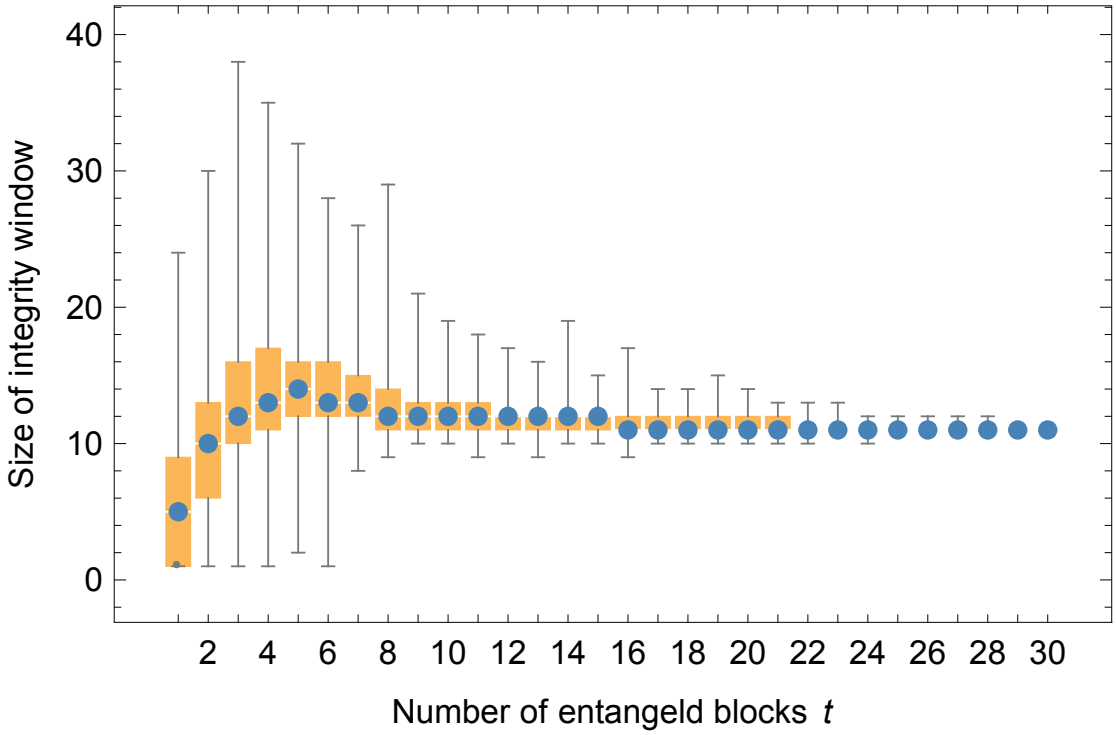
2.7.2 Regular entanglement within a sliding window

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

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

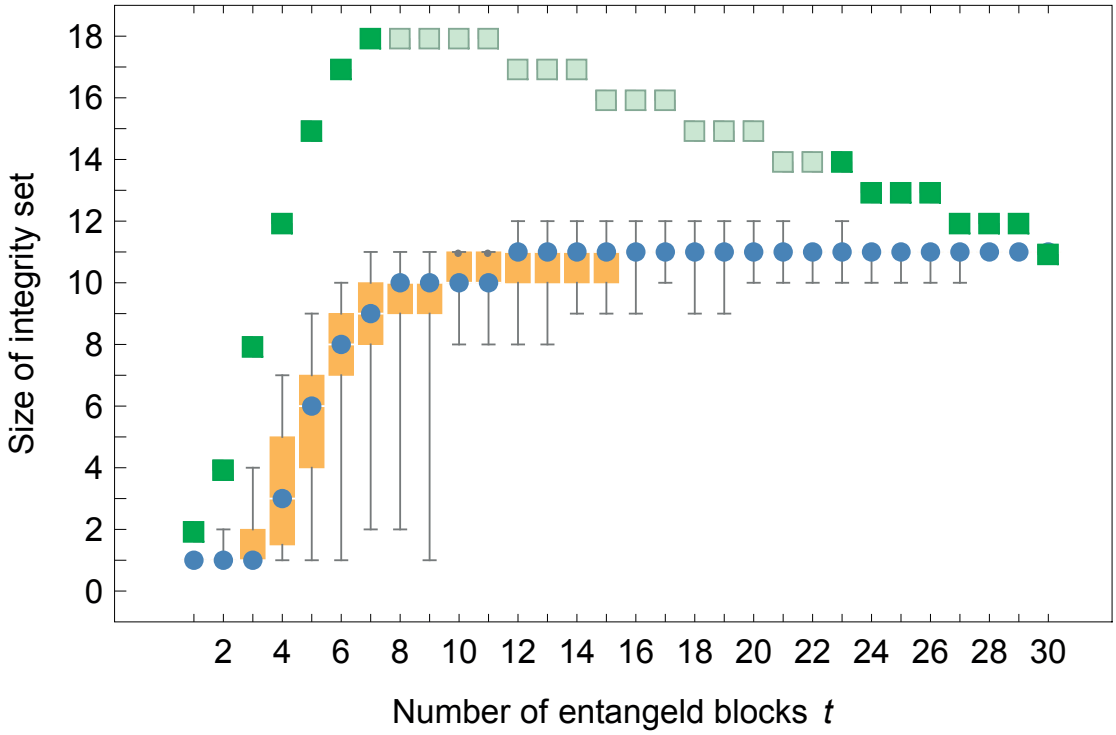


(a) Size of the smallest integrity set $I_{\min}(d_{1000})$. The green squares represent the size of the smallest integrity sets for the best regular entanglement within the window (see Section 2.7.2).

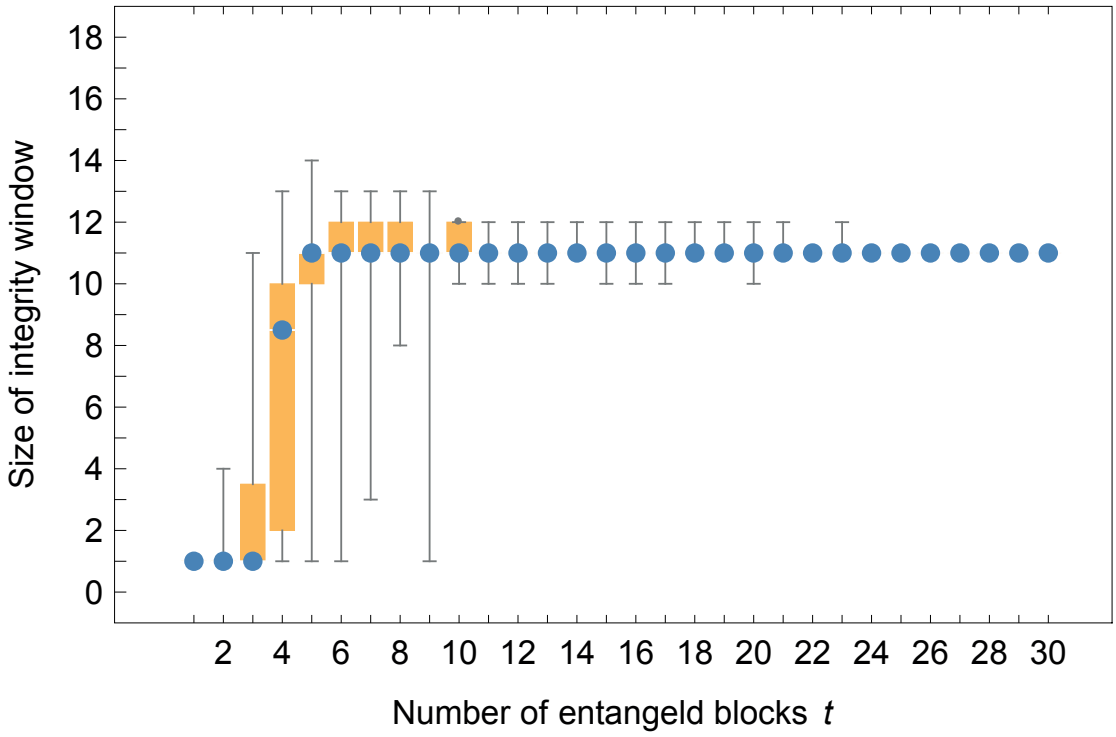


(b) Size of the integrity windows for the integrity sets found in Figure 2.8a.

Figure 2.8 – 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, 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.

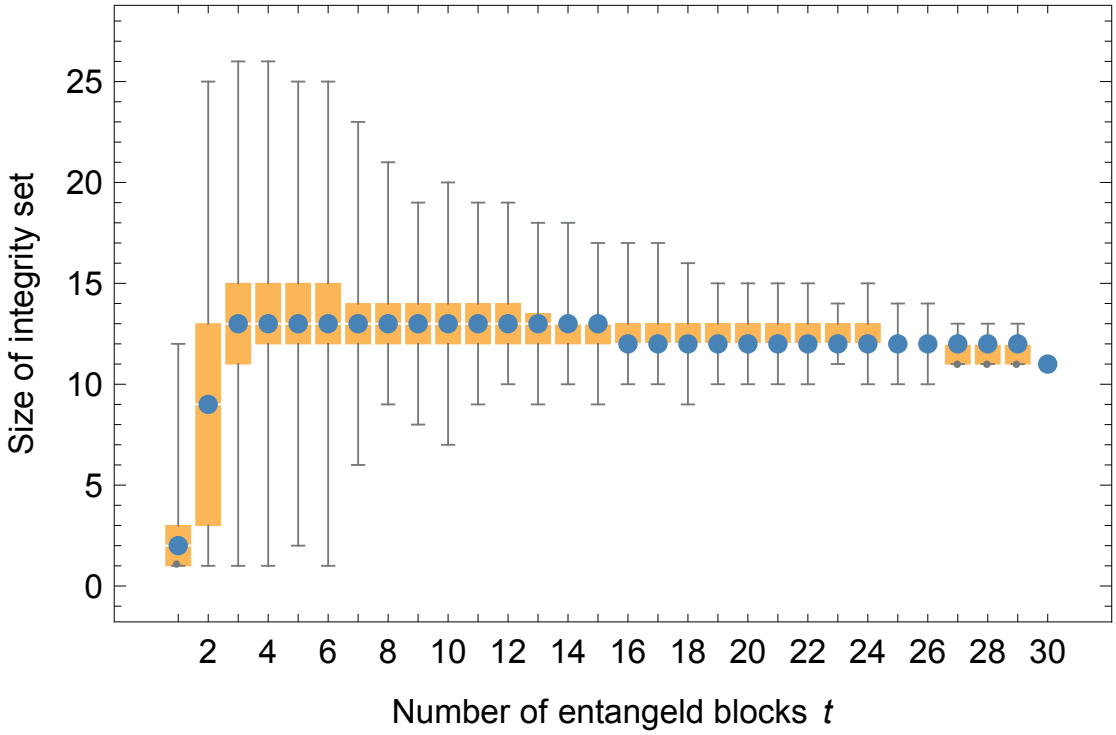


(a) Distribution of $\min_{990 \le k \le 1010} I_{\min}(d_k)$. The green squares represent the size of the smallest integrity sets for the best regular entanglement within the window (see Section 2.7.2).

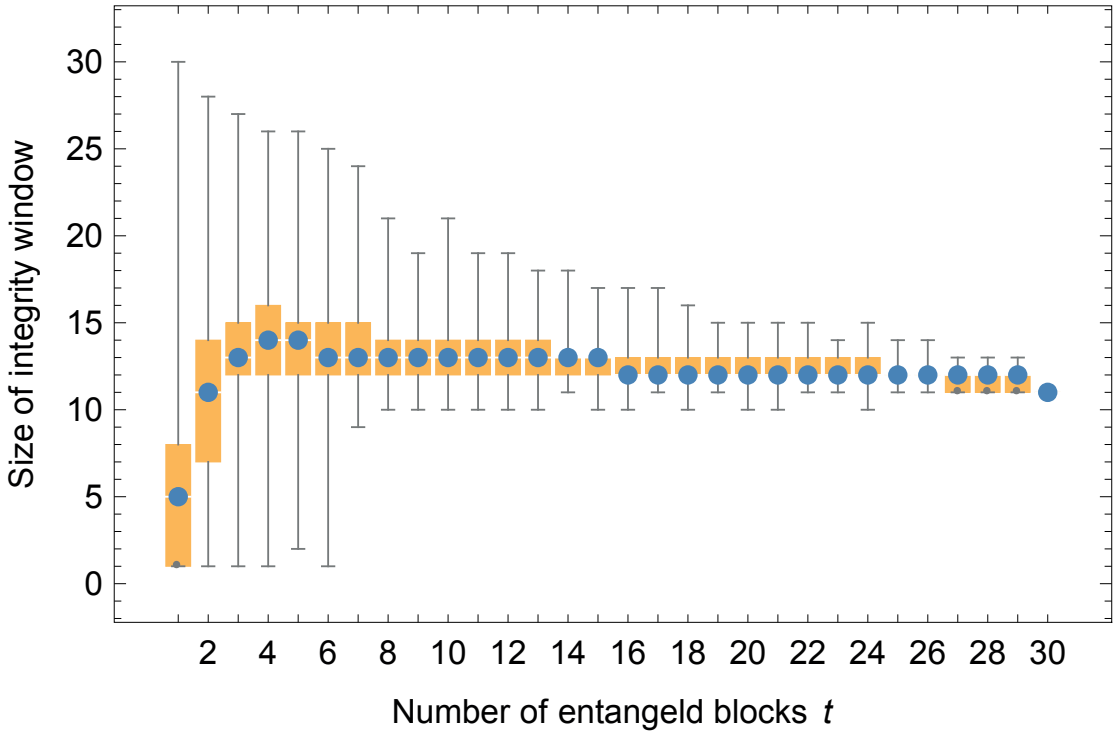


(b) Size distribution of the sliding windows for the integrity sets found in Figure 2.9a.

Figure 2.9 – 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.

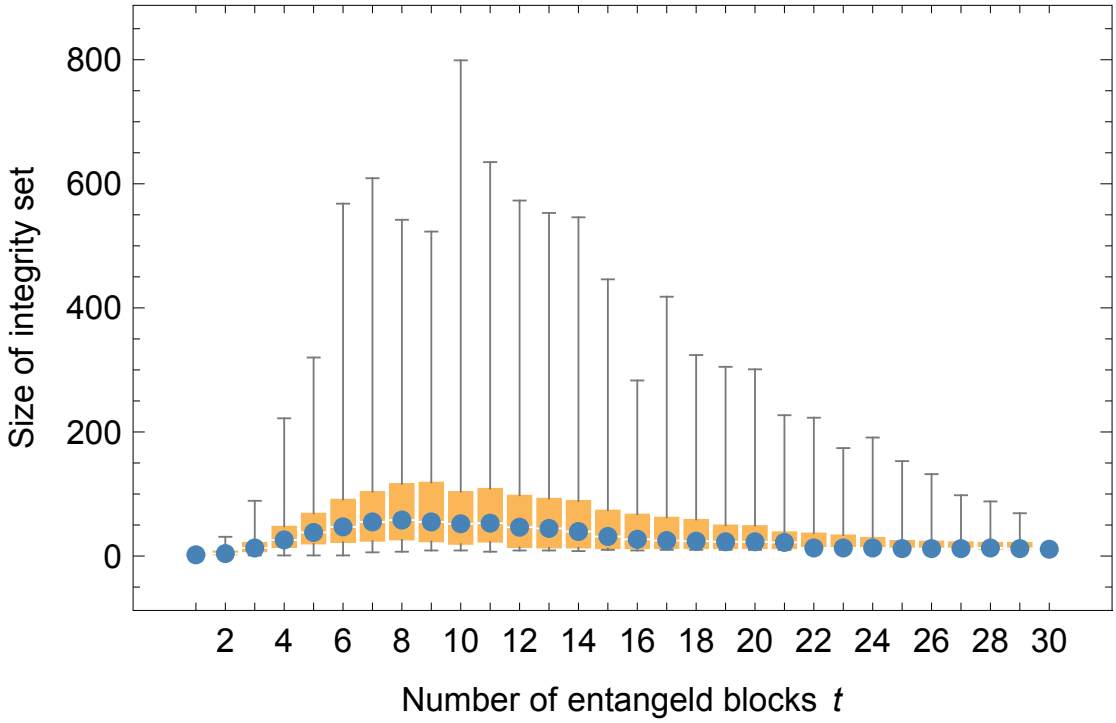


(a) Size of the smallest integrity set $I_{\min}(d_{1000})$.

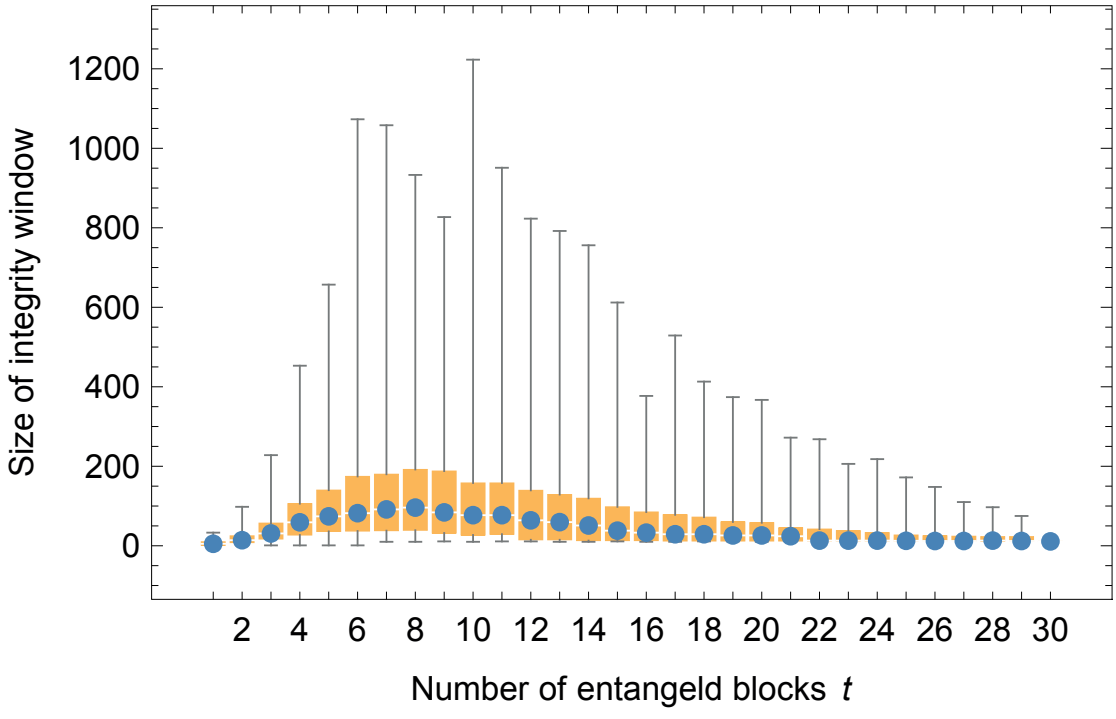


(b) Size of the integrity windows $W(d_{1000})$ for the integrity sets found in Figure 2.10a.

Figure 2.10 – 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 2.11a.

Figure 2.11 – 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}.

instance, with $t = 3$ pointers and a sliding window of size $w = 10$, the regular structure

$$(t_k^1 \triangleq b_{k-1}^1, t_k^2 \triangleq b_{k-3}^2, t_k^3 \triangleq b_{k-10}^3) \tag{2.8}$$

gives ${}_{\max}\mathscr{I}_{\min} = 8$, 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 2.8a and 2.9a 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 are lower bounds since we did not explore all the possible regular structures for the corresponding 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.

2.8 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 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, although the proof has eluded us so far.

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*

$$\begin{aligned} &E[L_i] = \infty && \text{if } t = 1; \\ &E[L_i] \sim k\left(1+\tfrac{1}{t-1}\right)^i && \text{if } t > 1 \text{ and } i \in o(k). \end{aligned} \tag{2.9}$$

Proof.

Case $i = 1$. We consider the random variable $L' \geq 1$ defined as

$$L' \triangleq L_1 - k. \tag{2.10}$$

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

$$\begin{aligned} \Pr[L' = l] &= \left(\frac{k-1}{k}\right)^t \cdot \left(\frac{k}{k+1}\right)^t \cdot \ldots \cdot \left(\frac{k+l-3}{k+l-2}\right)^t \\ &\qquad \cdot \left(1 - \left(\frac{k+l-2}{k+l-1}\right)^t\right) \\ &= \left(\frac{k-1}{k+l-2}\right)^t \left(1 - \left(\frac{k+l-2}{k+l-1}\right)^t\right) \\ &= (k-1)^t \cdot \left(\frac{1}{(k+l-2)^t} - \frac{1}{(k+l-1)^t}\right) \end{aligned} \tag{2.11}$$

The expectation of this random variable is

$$\begin{aligned} E[L'] &= \sum_{l=1}^{\infty} l \cdot \Pr[L' = l] \\ &= (k-1)^t \sum_{l=1}^{\infty} \left(\frac{l}{(k+l-2)^t} - \frac{l}{(k+l-1)^t}\right) \\ &= (k-1)^t \sum_{l=k-1}^{\infty} l^{-t}. \end{aligned} \tag{2.12}$$

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

$$\begin{aligned} (k-1)^t \int_{k-1}^{\infty} l^{-t}\, dl &\leq E[L'] \\ &\leq (k-1)^t \int_{k-1}^{\infty} (l-1)^{-t}\, dl \\ -(k-1)^t \cdot \frac{(k-1)^{-t+1}}{-t+1} &\leq E[L'] \leq -(k-1)^t \cdot \frac{(k-2)^{-t+1}}{-t+1} \\ \frac{k-1}{t-1} &\leq E[L'] \leq \frac{k-1}{t-1} \cdot \left(\frac{k-1}{k-2}\right)^{t-1}. \end{aligned} \tag{2.13}$$

Using (2.13), we obtain

$$E[L'] \sim \frac{k}{t-1} \tag{2.14}$$

and from (2.10) we can conclude that

$$E[L_1] \sim k\left(1+\frac{1}{t-1}\right). \tag{2.15}$$

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 $i = 1, 2, \ldots, n$ and prove the property for $n+1$.

Using the iterated expectation, we can write

$$\begin{aligned} E[L_{n+1}] &= E[E[L_{n+1} \mid L_1]] \\ &= \sum_{l_1 \geq k+1} \Pr[L_1 = l_1] \cdot E[L_{n+1} \mid L_1 = l_1]. \end{aligned} \tag{2.16}$$

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

$$E[L_{n+1}] = \sum_{l_1 \geq k+1} \Pr[L_1 = l_1] \cdot \left[l_1 \cdot \left(1+\frac{1}{t-1}\right)^n + o(k) \right]. \tag{2.17}$$

As done in (2.10) for the case $i = 1$, we use the random variable $L' = L_1 - k$ and rewrite (2.17) as

$$\begin{aligned} E[L_{n+1}] &= \sum_{l \geq 1} \Pr[L' = l] \cdot \left[(l+k) \cdot \left(1+\frac{1}{t-1}\right)^n + o(k) \right] \\ &= \left(1+\frac{1}{t-1}\right)^n E[L'] \\ &\quad + \left[k\left(1+\frac{1}{t-1}\right)^n + o(k) \right] \sum_{l \geq 1} \Pr[L' = l]. \end{aligned} \tag{2.18}$$

where $\Pr[L' = l]$ is defined as in (2.11).

From (2.14), the first term of (2.18) can be written as

$$\left(1+\frac{1}{t-1}\right)^n E[L'] = \left(1+\frac{1}{t-1}\right)^n \frac{k}{t-1} + o(k) \tag{2.19}$$

whereas from (2.11) the second term of (2.18) can be written as

$$\begin{aligned}\left[k\left(1+\frac{1}{t-1}\right)^n+o(k)\right]\sum_{l\geq 1}\Pr[L'=l]&\\ =\left[k\left(1+\frac{1}{t-1}\right)^n+o(k)\right](k-1)^t&\\ \cdot\sum_{l=1}^{\infty}\left(\frac{1}{(k+l-2)^t}-\frac{1}{(k+l-1)^t}\right)&\\ =k\left(1+\frac{1}{t-1}\right)^n+o(k).&\end{aligned} \tag{2.20}$$

Putting (2.19) and (2.20) together, we can conclude that

$$\begin{aligned}E[L_{n+1}]&=\left(1+\frac{1}{t-1}\right)^n\frac{k}{t-1}+k\left(1+\frac{1}{t-1}\right)^n+o(k)\\ &=k\left(1+\frac{1}{t-1}\right)^n\left(\frac{1}{t-1}+1\right)+o(k)\\ &=k\left(1+\frac{1}{t-1}\right)^{n+1}+o(k)\\ &\sim k\left(1+\frac{1}{t-1}\right)^{n+1}.\end{aligned} \tag{2.21}$$

□

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 an (s,t,e,p)-archive are chosen uniformly at random, then*

$$\begin{aligned}&E[M_i]=\infty && \text{if } t\leq p;\\ &E[M_i]\sim k\left(1+\tfrac{p}{t-p}\right)^i && \text{if } t>p \text{ and } i\in o(k).\end{aligned} \tag{2.22}$$

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

$$\begin{aligned} E[M_1] &\sim k\left(1+\frac{1}{t'-1}\right)^i \\ &\sim k\left(1+\frac{1}{\frac{t}{p}-1}\right)^i \\ &\sim k\left(1+\frac{p}{t-p}\right)^i. \end{aligned} \tag{2.23}$$

□

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*

$$E[N_l] \sim \ln\left(1+\frac{l}{k}\right)^t. \tag{2.24}$$

Proof. 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

$$1-\left(\frac{m+k-2}{m+k-1}\right)^t, \tag{2.25}$$

thus

$$\begin{aligned} E[N_l] &= \sum_{m=1}^{l}\left(1-\left(\frac{m+k-2}{m+k-1}\right)^t\right) \\ &= l-\sum_{m=1}^{l}\left(\frac{m+k-2}{m+k-1}\right)^t \\ &= l-\sum_{m=1}^{l}\sum_{j=0}^{t}\binom{t}{j}\cdot\frac{(-1)^j}{(m+k-1)^j} \\ &= l-\sum_{m=1}^{l}\left(1-\frac{t}{m+k-1}+o(k^{-1})\right). \end{aligned} \tag{2.26}$$

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

$$\begin{aligned} E[N_l] &= l-l+t\sum_{m=1}^{l}\frac{1}{m+k-1}+o(k) \\ &= t(H_{l+k-1}-H_k) \end{aligned} \tag{2.27}$$

where H_n is the n-th partial sum of the harmonic series, which can be bounded by $H_n =$

$\ln n + o(n)$ [GKP94]. We can therefore conclude that

$$\begin{aligned} E[N_l] &= t\,(\ln(l+k-1) - \ln(k)) + o(k) \\ &\sim t \ln\left(1 + \frac{l}{k}\right). \end{aligned} \tag{2.28}$$

□

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*

$$D_i \sim k \cdot \left(e^{\frac{1}{t}}\right)^i. \tag{2.29}$$

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

$$l \sim k(e^{\frac{i}{t}} - 1), \tag{2.30}$$

thus

$$\begin{aligned} D_i &= k + l \\ &\sim k + k(e^{\frac{i}{t}} - 1) \\ &= k\left(e^{\frac{1}{t}}\right)^i. \end{aligned} \tag{2.31}$$

□

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 file 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 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 file permanently by deleting a sublinear number of other files 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. We split the leaping attack into levels $1,2,3,\ldots$ 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

$$k_n = k\left(e^{\frac{1}{t}}\right)^n. \tag{2.32}$$

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

$$x_n = x_{n-p+e-1} + \cdots + x_{n-p} + \frac{1}{p}\sum_{i=1}^{n-p-1} x_i. \tag{2.33}$$

To solve this recurrence relation, we can write

$$x_{n-1} = x_{n-p+e-2} + \cdots + x_{n-p-1} + \frac{1}{p}\sum_{i=1}^{n-p-2} x_i, \tag{2.34}$$

and by subtracting (2.34) from (2.33) it follows that

$$\begin{aligned} x_n - x_{n-1} &= x_{n-p+e-1} + \frac{1}{p}x_{n-p-1} - x_{n-p-1} \\ x_n &= x_{n-1} + x_{n-p+e-1} + \left(\frac{1}{p} - 1\right)x_{n-p-1}. \end{aligned} \tag{2.35}$$

The solution of the recurrence relation is

$$x_n = C_1 r_1^n + C_2 r_2^n + \cdots + C_{p+1} r_{p+1}^n \tag{2.36}$$

where the C_i are constants and the r_i are the roots of the characteristic polynomial[9]

$$f(x) = x^{p+1} - x^p - x^e + 1 - \frac{1}{p}. \tag{2.37}$$

The rate of growth of x_n when n is large is

$$\begin{aligned} \lim_{n\to\infty} \frac{x_{n+1}}{x_n} &= \frac{C_1 r_1^{n+1} + C_2 r_2^{n+1} + \cdots + C_{p+1} r_{p+1}^{n+1}}{C_1 r_1^n + C_2 r_2^n + \cdots + C_{p+1} r_{p+1}^n} \\ &= r \end{aligned} \tag{2.38}$$

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 $E[I_{\min}(d_k)] \in o(K)$. □

It should be mentioned that the recurrence relation (2.33) is not optimal for three reasons. Firstly, it assumes that there are no collisions, i.e., that all the documents at level n are different. Pointers that collide are advantageous for the attacker because the involved documents can be destroyed by deleting less than e blocks from them (they have more than one erased tangled pointer block). Secondly, it assumes that an attacker always targets parity blocks when completing the deletion of a file. 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 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 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 $E[I_{min}(d_k)] \in o(K)$.*

Proof. 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

[9] 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.

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

$$\Pr[M = m] = \left(\frac{p-1}{p}\right)^m. \tag{2.39}$$

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

$$x_n(m) = x_{n-m-1} + \frac{1}{p}\sum_{i=1}^{n-m-2} x_i. \tag{2.40}$$

The asymptotic recurrence relation for the leaping attack is therefore

$$\begin{aligned} x_n &= \sum_{m=1}^{\infty} \Pr[M = m] \cdot x_n(m) \\ &= \frac{p-1}{p}\sum_{n=1}^{n-2} x_i. \end{aligned} \tag{2.41}$$

To solve this recurrence relation, we can write

$$x_{n-1} = \frac{p-1}{p}\sum_{n=1}^{n-3} x_i \tag{2.42}$$

and by subtracting (2.42) from (2.41) it follows that

$$x_n = x_{n-1} + \frac{p-1}{p}x_{n-2}. \tag{2.43}$$

The solution of the recurrence relation is

$$x_n = c_1\left(\frac{1+\sqrt{5-\frac{4}{p}}}{2}\right)^n + c_2\left(\frac{1-\sqrt{5-\frac{4}{p}}}{2}\right)^n \tag{2.44}$$

and rate of growth of x_n when n is large is

$$r \triangleq \lim_{n\to\infty}\frac{x_{n+1}}{x_n} = \frac{1+\sqrt{5-\frac{4}{p}}}{2}. \tag{2.45}$$

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 $E[I_{\min}(d_k)] \in o(K)$. □

Table 2.1 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

		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

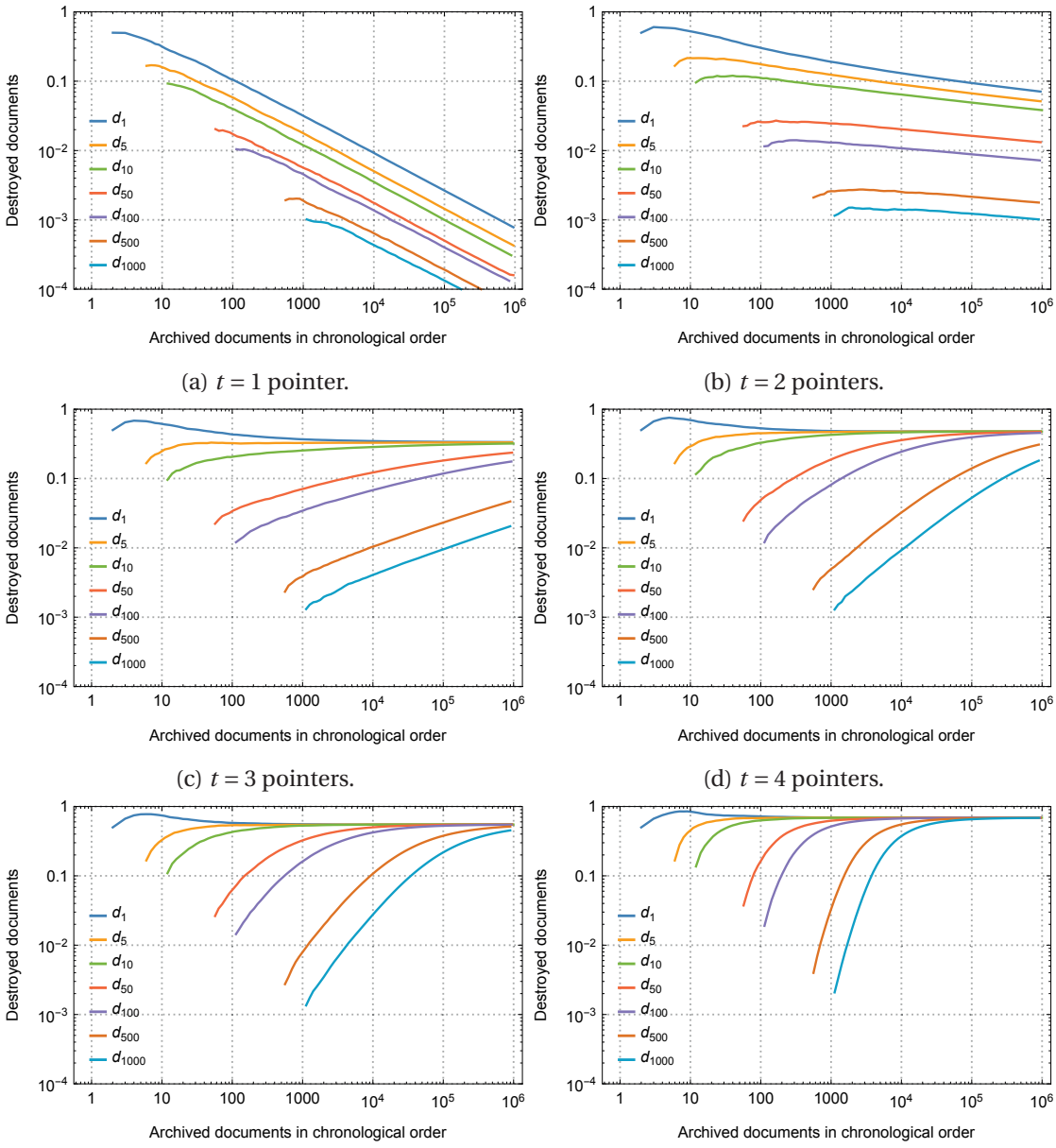
Table 2.1 – Lower bound for the number of pointers from Theorem 16 and Lemma 17 for different code rates $\frac{s}{p}$.

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 2.12. 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 2.12. Increasing t further accelerates the convergence and increases the fraction of documents that must be destroyed.

2.8.1 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 2.6.3. 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 15) and tailored (Section 2) 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



(a) $t = 1$ pointer.

(b) $t = 2$ pointers.

(c) $t = 3$ pointers.

(d) $t = 4$ pointers.

(e) $t = 5$ pointers.

(f) $t = 10$ pointers.

Figure 2.12 – 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.

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 2.13 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 2.14 shows the damage caused by the minimum bounded breadth-first attack with the same simulation parameters. The figures provide numerical evidence supporting our conjecture and show the efficiency of the greedy leaping attack

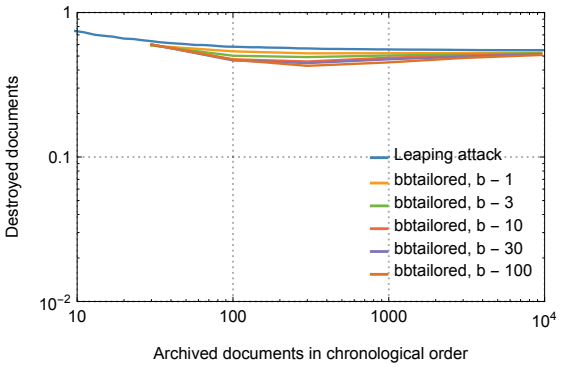
2.9 Extensions and Discussion

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

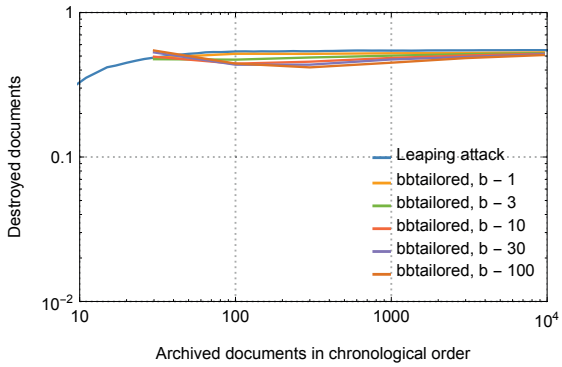
2.9.1 To store or not to store source blocks ?

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.

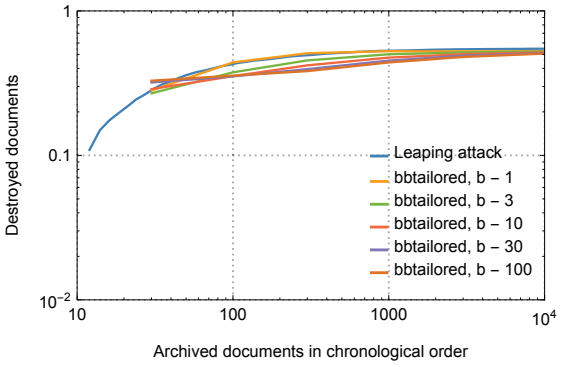
It is tempting to remove this overhead by including the source blocks in the systematic form of the code, and allow the clients to directly fetch the source blocks. However, from the censorship-resistance perspective, a systematic code would invalidate all our previous reasoning. We can no longer equate 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 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 can thus only provide *weak censorship-resistance*, as this attack, even at a high material cost, does not affect data availability.



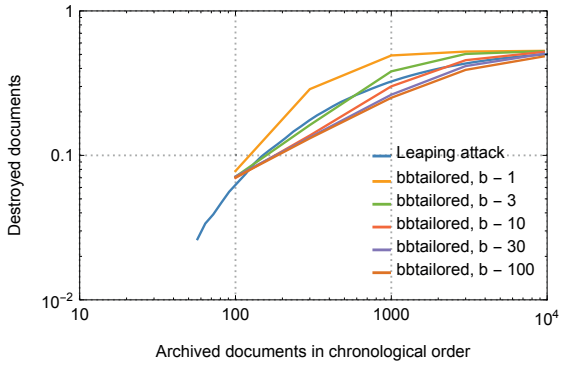
(a) Target document d_1.



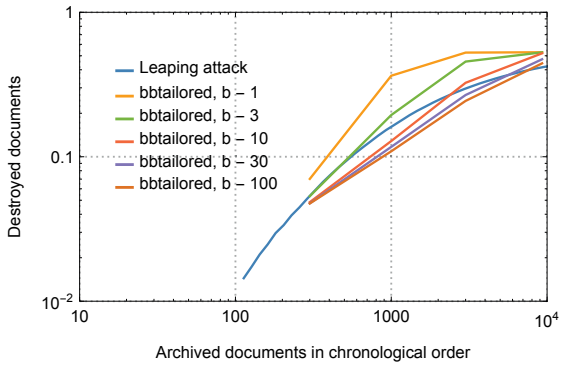
(b) Target document d_5.



(c) Target document d_{10}.

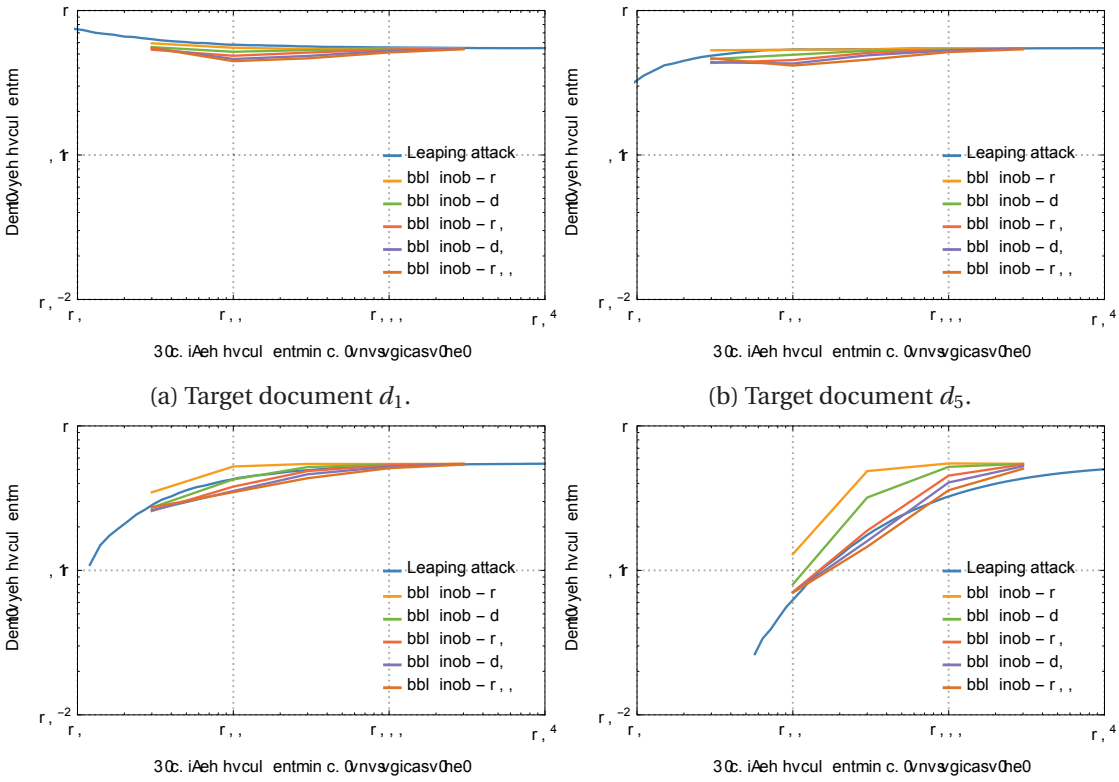


(d) Target document d_{50}.



(e) Target document d_{100}.

Figure 2.13 – 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}.

Figure 2.14 – 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.

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 complexity tradeoff in terms of computational effort versus number of blocks to make the attack irrecoverable.

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 block 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 [WOW08]. 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.

2.9.2 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. 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 [WOW08], 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.

2.9.3 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 older documents. Furthermore, the former document must be older than all the latter. 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 [Tar76]. 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.

2.9.4 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 code words $c_{k+1}, \ldots, c_n$ 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, for which no parity blocks are ever pointed. 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.

2.9.5 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 or stored locally by the clients interested in a document; or, at the very least, that clients can store a cryptographically secure fingerprint of the relevant metadata. 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, erases $e+1$ blocks from the document codeword, censors the document by replacing its content with something else, 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. 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 $|S| \leq E_{\min}(d_k)$. 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 $|S| \leq E_{\min}(d_k)$.

We now show that $|S| \geq E_{\min}(d_k)$. 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 $E_{\min}(d_k) \leq |S|$. □

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.

2.9.6 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.).

These techniques are applicable to fundamentally different scenario 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).

2.9.7 Space reclaiming

A very serious restriction of our model is the apparent inability to delete past files. Implementing a deletion mechanism is not trivial, as it should prevent attackers from using it to perform censorship. In the setting of immutable storage, data has no canonical owner, and it is not always possible to distinguish a "legitimate owner" from an attacker, assuming a reliable definition of legitimate owner is even possible.

With random entanglement, protection increases over time. Such a system cannot reclaim space from old unwanted files, and is entirely dependant on the storage system being able to grow as fast as its users require, while retaining everything forever. Clearly, this is not sustainable.

In the case of proximity entanglement, as we have shown, it is possible to destroy the tail of the archive without losing current data. To solve the problem of space reclaiming, an intermediary solution would involve a STEP-archive rolling in time, with parity blocks being garbage-collected in order at a predictible time. This system would require clients to periodically refresh their data, by recreating new versions (with different metadata) of their files at fixed intervals. Obsolete files would naturally disappear after a while, if no user took responsibility for refreshing them. Randomized convergent encryption would be required to prevent a censor for recognizing the new blocks immediately, and blocking their insertion into the system.

2.10 Conclusion and Open Problems

In this chapter, we introduced and studied STEP-archives with the objective of providing strong 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 do so there are theoretical and practical questions to explore.

2.10.1 Variable block size

Imposing a fixed block size at the system level is rigid and inefficient. An unnecessary large block size wastes storage space, decoding time and bandwidth, but a too small block size bloats metadata and increases the number of IO 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.

2.10.2 Data expiration

We currently 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(s) to drive deletion would make them designated targets for a determined censor. 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 2.8, do not allow this. But in principle, if bounded distributions like the sliding windows used in section 2.7 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 they expire. 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 file[10]. 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 [ZX12, JKJ07] mechanism allowing a client to interactively query the storage system, to ensure with high probability the legitimacy of a new metadata block.

[10] 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.

2.10.3 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 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. Solutions in this direction will probably involve a mix of partly structured pointers to the recent past and pointers arbitrarily far in the past. Another path worth exploring is to use STEP-archival in parallel with regular replication for new documents and to get rid of the replicas as they age.

Finally, while we focused on MDS codes in this chapter, 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.

3 Public Keys

3.1 Introduction

We performed a sanity check of public keys collected on the web and found that the vast majority works as intended. Our main goal was to test the validity of the assumption that different random choices are made each time keys are generated. We found that this is not always the case, resulting in public keys that offer no security. Our conclusion is that generating secure public keys in the real world is challenging. We did *not* study usage of public keys.

Various studies have been conducted to assess the state of the current public key infrastructure, with a focus on X.509 certificates (cf. [CSF+08]). Key generation standards for RSA (cf. [RSA78]) have been analysed and found to be satisfactory in [LN11]. In [HBKC11] and [VFBH11] (and the references therein) several problems have been identified that are mostly related to the way certificates are used. In this chapter we complement previous studies by concentrating on computational and randomness properties of actual public keys, issues that are usually taken for granted.

Compared to the collection of certificates considered in [HBKC11], where shared public keys are "not very frequent", we found a much higher fraction of duplicates. We also found public keys that are not related to the Debian OpenSSL vulnerability but that offer no security at all. The existence of such keys may be crypto-folklore, but it was new to us (but see [Joh99]). This is not a disappearing trend, as may be seen by comparing the results in this chapter to those reported in [LHA+12b]. Vulnerabilities of this sort could affect the expectation of security that the public key infrastructure is intended to achieve. We limited our study to collections of public keys and did not consider issues arising while using them.

We summarize our findings, referring to later sections for details. We collected as many openly accessible public keys as possible from the web, while avoiding activities that our system administrators may have frowned upon. In particular we did not capture or analyse any encrypted traffic of digitally signed documents. The set of 11.7 million public keys that we collected contains 6.4 million distinct RSA moduli. The remainder is almost evenly split between

ElGamal keys (cf. [ElG85]) and DSA keys (cf. [U.S09]), plus a single ECDSA key (cf. [U.S09]). The frequency of keys blacklisted due to the Debian OpenSSL vulnerability (cf. [YRS+09]) is comparable to [HBKC11]; the findings presented below are not related to this vulnerability. All keys were checked for consistency such as compositeness, primality, and (sub)group membership tests. As the sheer number of keys and their provenance precluded extensive cryptanalysis and the sensibility thereof, per key a modest search for obvious weaknesses was carried out as well. These efforts resulted in a small number of inconsistent or weak keys.

A tacit and crucial assumption underlying the security of the public key infrastructure is that during key setup previous random choices are not repeated. In [HBKC11, LN11] public key properties are considered but this issue is not addressed, with [LN11] nevertheless concluding that

> The entropy of the output distribution [of standardized RSA key generation] is always almost maximal, ... and the outputs are hard to factor if factoring in general is hard.

We do not question the validity of this conclusion, but found that it can only be valid if each output is considered in isolation. When combining outputs the above assumption sometimes fails. Among all types of public keys collected (except ECDSA), we found duplicates with unrelated owners. This is a concern because, if these owners find out, they may breach each other's security. Duplication of keys is more frequent in our collection than in the one from [HBKC11].

We also stumbled upon RSA moduli, not affected by the Debian OpenSSL vulnerability, that offer no security. Their secret keys are accessible to anyone who redoes our work. Assuming access to the public key collection, this is straightforward compared to more traditional ways to retrieve RSA secret keys (cf. [Cop93, LL93]).

Figure 3.1 depicts a simplified sketch of the situation and how it may evolve.

An existing collection of seven (black) keys is extended with six (red) new keys, where capital letters play the role of (matching) large primes. Initially, keys AB, CD, EF, GH, and JK on the left are secure and keys LM and LN on the right are openly insecure in the same keyring due to the common factor L. New key PQ is secure and appended to the secure list on the left. New key AB duplicates key AB on the left, making both insecure to each other but not to anyone else. New key LM duplicates a key already known to be in the openly insecure group, while key LR results in a new openly insecure modulus on that keyring. Key ES removes known good key EF from the secure keys on the left, resulting in a new openly insecure group on the right consisting of keys EF and ES. Even if the owner of ES now knows that he is insecure and destroys the key, this information can be used by any owners involved to determine the factors of key EF. Key GJ removes two known good keys, GH and JK, from the list of secure keys on the left to form an insecure double keyring on the right (cf. Figure 3.5 in Section 3.3). All example

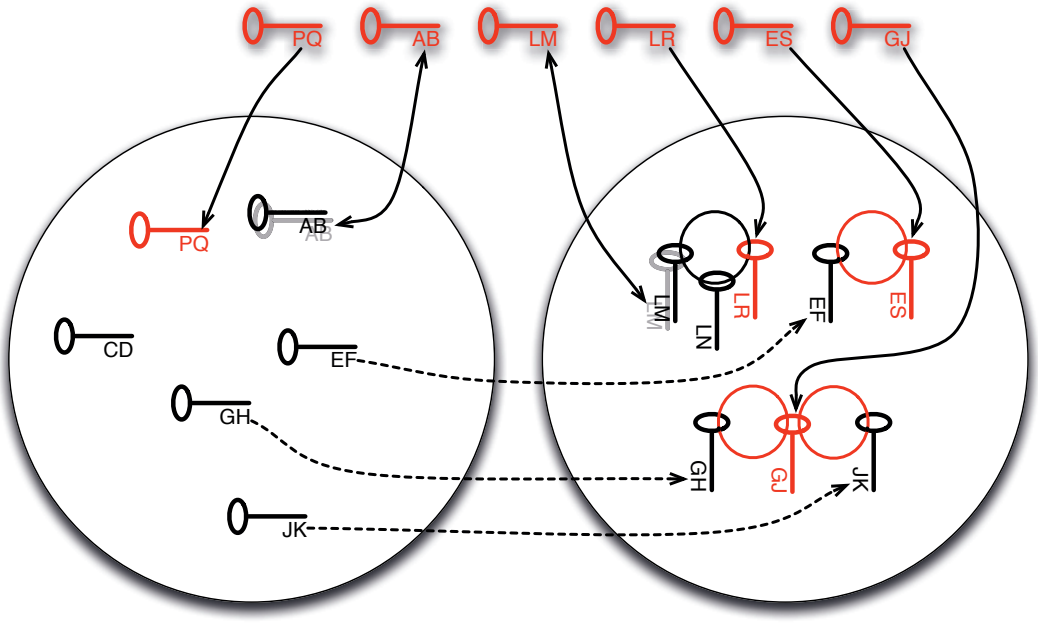


Figure 3.1 – Illustration of attack scenarios

keyrings, and many more, occur in the real world. Note that a key that has been dragged from left to right will never be able to return.

Our findings, of which we do not and will not publicly present any evidence, are confirmed by independent similar work (cf. [Hen12]). As shown in Section 3.3 we used a computational method different from the one used in [Hen12].

Section 3.2 presents our data collection efforts. Sections 3.3 and 3.4 describe the counts and calculations performed for the RSA-related data and for the ElGamal, DSA, and ECDSA data, respectively. Section 3.5 summarizes our findings.

3.2 Data collection

Before the data from [Ele10] was generally available, we started collecting public keys from a wide variety of sources, assisted by colleagues and students. We collected *only* public keys, no encrypted data or digitally signed documents (other than digital certificates). This resulted in almost 5.5 million PGP keys and fewer than 0.1 million X.509 certificates. The latter got a boost with [Ele10] and, to a smaller extent, the data from [HBKC11]. We did not engage in web crawling, extensive `ssh`-session monitoring, or other data collection activities that may be perceived as intrusive, aggressive, or unethical. Thus, far more data can be collected than we did (see also [LHA+12b]).

Skipping a description of our attempts to agree on a sufficiently uniform and accessible representation of the data, by November 2011 the counts had settled as follows: 6 185 372 distinct X.509 certificates (most from the EFF SSL repository, 43 from other sources), and 5 481 332 PGP keys, for a total of at most 11 666 704 public keys. Of the X.509 certificates 6 185 230 are labeled to contain an RSA (modulus, exponent) pair with 141 DSA public keys and a single ECDSA point on the NIST standardized curve `secp384r1` (cf. [Cer00, Section 2.8], [U.S09]). Of the certificates 47.6% have an expiration date later than 2011. About 77.7% of the certifying signatures use SHA1 or better (5287×SHA256, 24×SHA384, 525×SHA512), 22.3% use MD5 (with 122×MD2, 30×GOST, 14×MD4, and 9×RIPEMD160). Both requirements, expiration later than 2011 and usage of SHA1-or-2, are met by 33.4% of the certificates.

Of the PGP keys 2 546 752 (46.5%) are labeled as ElGamal public keys, 2 536 959 (46.3%) as DSA public keys, the other 397 621 (7.3%) as RSA public keys. PGP keys have no expiration dates or hashes. All public keys were further analysed as described below.

3.3 RSA

In this section we present the results of various counts and tests that we conducted on the data labeled as RSA public keys. An RSA public key is a pair (n, e) of a supposedly hard to factor RSA modulus n and a public exponent e. The corresponding secret key is the integer d such that $de \equiv 1 \bmod \varphi(n)$ or, equivalently, the factorization of n.

Public exponents. Table 3.1 lists the ten most frequent public exponents along with their percentage of occurrence for the RSA keys in the X.509 certificates, the PGP keys, and when combined. Except for eight times $e = 1$ and two even e-values among the PGP RSA keys, there is no reason to suspect that the e-values are not functional. Two e-values were found that, due to their size and random appearance, may correspond to a short secret exponent (we have not investigated this). The public exponents are not further regarded below.

Debian moduli. Two of the n-values, a 1024 and a 2048-bit one each occurring once, were discarded because they could be fully factored using the data from [Moo08] (cf. Debian OpenSSL vulnerability in [YRS+09]). A further 30097 n-values (0.48%, with 21459 distinct ones) were found to be blacklisted (cf. [T+]), but as their factors were not easily available they were kept.

Shared moduli. We partition the set of 6 185 228 X.509 certificates into *clusters*, where certificates in the same cluster contain the same RSA modulus. There is a considerable number of clusters containing two or more certificates, each of which could be a security issue; clusters consisting of one certificate, on the other hand, are the good cases. As depicted in Figure 3.2, there is one cluster with 16489 certificates (the blue circle on the x-axis), followed by clusters of sizes 8366, 6351, 5055, 3586, 3538, 2645, for a total of 14 clusters with more than a thousand certificates (the red and blue circles on the x-axis; the 5055 share a blacklisted modulus,

Table 3.1 – Most frequently occurring RSA public exponents.

X.509		PGP		Combined	
e	%	e	%	e	%
65537	98.4921	65537	48.8501	65537	95.4933
17	0.7633	17	39.5027	17	3.1035
3	0.3772	41	7.5727	41	0.4574
35	0.1410	19	2.4774	3	0.3578
5	0.1176	257	0.3872	19	0.1506
7	0.0631	23	0.2212	35	0.1339
11	0.0220	11	0.1755	5	0.1111
47	0.0101	3	0.0565	7	0.0596
13	0.0042	21	0.0512	11	0.0313
65535	0.0011	$2^{127}+3$	0.0248	257	0.0241
other	0.0083	other	0.6807	other	0.0774

with no other blacklisted modulus occurring more than seven times). On the other side of the scale the number of good cases is 5 918 499 (the single green circle on the y-axis), with 58913 and 7108 clusters consisting of two and three certificates, respectively. It follows that $6\,185\,228 - 5\,918\,499 = 266\,729$ X.509 certificates (4.3%) contain an RSA modulus that is shared with another X.509 certificate. With 71024 clusters containing two or more certificates it follows that there are $5\,918\,499 + 71024 = 5\,989\,523$ different n-values.

Looking at the owners with shared n-values among the relevant set of 266 729 X.509 certificates, many of the duplications are re-certifications or other types of unsuspicious recycling of the same key material by its supposedly legal owner. It also becomes clear that any single owner may come in many different guises. On the other hand, there are also many instances where an n-value is shared among seemingly unrelated owners. Distinguishing intentionally shared keys from other duplications (which are prone to fraud) is not straightforward, and is not facilitated by the volume of data we are dealing with (as 266 729 cases have to be considered). We leave it as a subject for further investigation into this "fuzzy" recognition problem to come up with good insights, useful information, and reliable counts.

The 397 621 PGP RSA keys share their moduli to a much smaller extent: one n-value occurs five times and 27 occur twice. Overall, 28 n-values occur more than once, for a total of 59 occurrences. The n-value that occurs in five PGP keys also occurs twice among the X.509 certificates, and all seven occurrences refer to the same owner. For some of the other 27 multiple occurrences of n-values unique ownership of the RSA keys was harder to assess.

Distinct moduli. As seen above, we extracted 5 989 523 different n-values from the X.509 certificates. Similarly, $397\,621 - 59 + 28 = 397\,590$ of the PGP n-values are unique. Joining the two sets resulted in 6 386 984 distinct values, with the 129 n-values contained in both sets occurring in 204 X.509 certificates and in 137 PGP keys: as mentioned, some PGP keys are X.509-certified as well (though we have not tried to establish unique or conflicting ownerships,

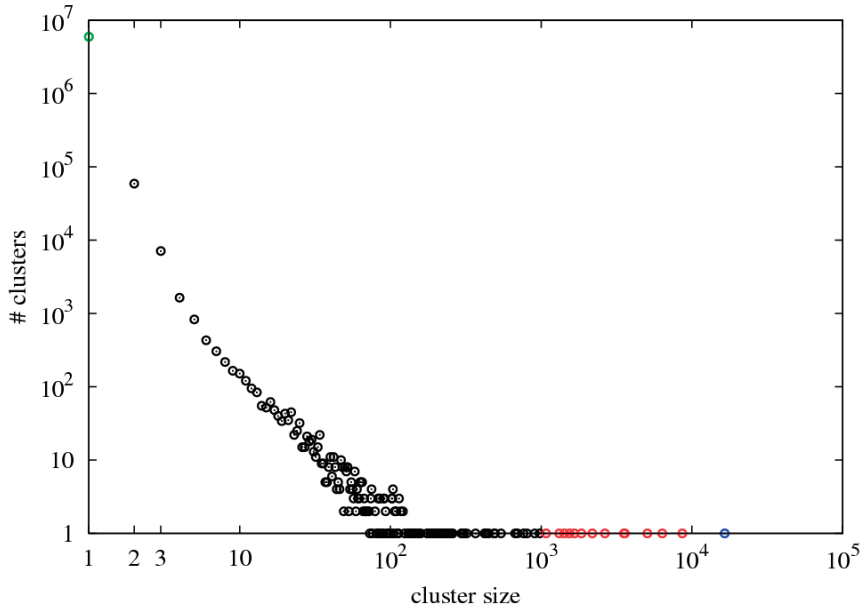


Figure 3.2 – Number of certificate clusters as a function of the cluster-size.

as this already proved to be infeasible for keys shared just among X.509 certificates). In order not to make it easier to re-derive our results, most information below refers to the joined set of unique values, not distinguishing between X.509 and PGP ones.

Modulus sizes. The cumulative sizes of the moduli in the set of 6 386 984 n-values are depicted in Figure 3.3. Although 512-bit and 768-bit RSA moduli were factored in 1999 (cf. [CDL+00]) and 2009 (cf. [KAF+10]), respectively, 1.6% of the n-values have 512 bits (with 0.01% of size 384 and smallest size 374 occurring once) and 0.8% of size 768. Those moduli are weak, but still offer marginal security. A large number of the 512-bit ones were certified after the year 2000 and even until a few years ago. With 73.9% the most common size is 1024 bits, followed by 2048 bits with 21.7%. Sizes 3072, 4096, and 8192 contribute 0.04%, 1.5%, and 0.01%, respectively. The largest size is 16384 bits, of which there are 181 (0.003%).

Primality, small factors, and other tests. Two of the unique n-values are prime, 171 have a factor $< 2^{24}$ (with 68 even n-values) after removal of which six cofactors are prime. About 25% of the remaining 165 composites were fully factored after a modest search for small factors using the implementation from [Zim12] of the elliptic curve method (ECM, cf. [Len87]), some of the others may indeed be hard to factor and could, in principle, serve as RSA modulus. Nevertheless, these 173 n-values do not comply with the standards for the generation of RSA moduli (cf. [LN11]) and they were discarded. Nine cases are probably due to copy-and-paste errors, as eight proper moduli were found that differed from the wrong ones in a few hexadecimal positions (two distinct wrong moduli match up with the same correct one).

Fermat's factorization method, which works well if two factors are close together, did not

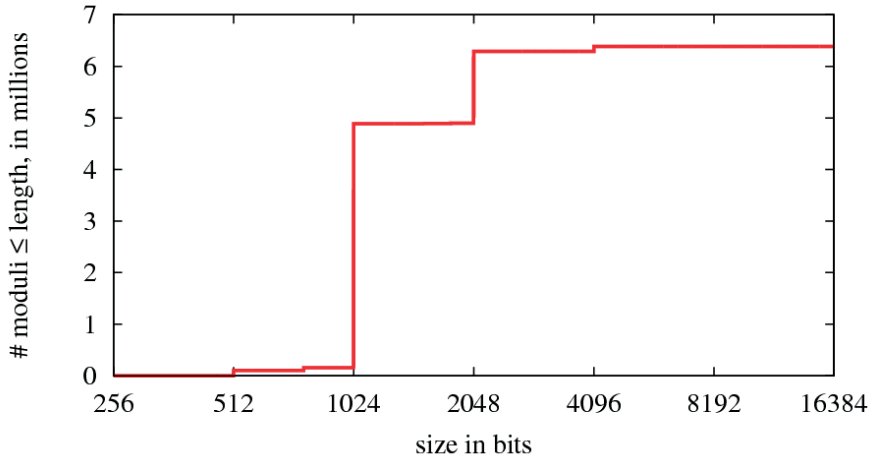


Figure 3.3 – Cumulative number of modulus sizes for RSA.

produce any factors. In particular we found no moduli as reported by Mike Wiener [Wie92] ($n = pq$ with p prime and q the least prime greater than p, and thus with p the largest prime $\leq [\sqrt{n}]$). Neither were there any non-trivial powers.

Moduli with shared factors. Moduli that share one prime factor result in complete loss of security for all moduli involved. We discuss the results based on the graphs spawned by the moduli and shared factors. If different users make different choices during key setup, the graph associated to c distinct n-values (cf. Introduction) would consist of c connected components[1], each consisting of a single edge connecting two unidentified – and supposedly unidentifiable – primes. This turned out not to be the case: it took a matter of hours on a single core to find 1995 connected components that each consist of at least two edges. Much larger datasets can be handled without trouble. Our calculation uses a simple-minded binary tree, forming a parent node $\mathrm{lcm}(a,b)$ for leaves a, b while taking appropriate action if $\gcd(a,b) > 1$ and using the subquadratic multiplication and greatest common divisor implementations from [Fre11]. It scales well and is marginally slower than the more contrived gcd-calculation described in [Hen12] but uses less memory. On a 1.8GHz i7 processor the straightforward approach would require about ten core-years and would not scale well. Inclusion of the p and q-values from Section 3.4 and the primes from [Moo08] related to the Debian OpenSSL vulnerability [YRS+09] did not produce additional results.

Of the 1995 connected components, 1988 are depth one trees[2]. Of those 1200 have two leaves (i.e., 1200 pairs of n-values, each with a distinct prime factor in common), 345 three leaves, etc., up to a single one with 4627 leaves (i.e., 4627 n-values all with the same prime factor in common). It is not uncommon for an n-value corresponding to an edge of these depth one

[1] Two distinct vertices are in the same connected component if and only if they are connected by a path consisting of edges in the graph.

[2] A depth one tree has no cycles and contains one *root* vertex with edges leading to all other vertices, as in Figure 3.4.

Table 3.2 – The s-column indicates the number of depth one trees with ℓ leaves for which all edge multiplicities are equal to one, the m-column the number of trees for which at least one edge occurs at least twice, and $T = s + m$ the total. The bold entry corresponds to the depth one tree depicted in Figure 3.4.

ℓ	s	m	T	ℓ	s	m	T	ℓ	s	m	T	ℓ	s	m	T
2	1009	191	1200	13	3	3	6	24	1	1	2	59	0	1	1
3	259	86	345	14	2	3	5	26	0	1	1	61	0	1	1
4	95	44	139	15	1	4	5	27	0	1	1	63	1	2	3
5	43	32	75	16	2	1	3	32	0	1	1	65	0	1	1
6	23	29	52	17	1	2	3	33	0	1	1	92	0	1	1
7	20	19	39	18	1	1	2	35	0	2	2	121	0	1	1
8	13	17	30	19	1	2	3	36	1	1	2	151	0	1	1
9	4	11	15	20	1	2	3	37	0	1	1	348	0	1	1
10	3	8	11	21	0	3	3	42	0	1	1	379	0	1	1
11	3	9	12	22	1	2	3	44	0	2	2	**4627**	0	**1**	**1**
12	3	3	6	23	0	1	1	46	0	1	1				

trees to occur more than once as an RSA modulus: 497 of the 1988 depth one trees have at least one edge that corresponds to an RSA modulus that occurs in at least two X.509 certificates or PGP keys. In the other 1491 depth one trees all edge multiplicities are one. Table 3.2 lists for each number of leaves ℓ how often each type occurs, with the s-column the number of trees for which all n-values occur once as RSA modulus in an X.509 certificate or PGP key, the m-column the number of trees for which at least one n-value occurs as RSA modulus in at least two X.509 certificates or PGP keys, and the total $T = s + m$. For smaller tree-sizes s is larger, for larger trees multiple occurrence of moduli is more common.

Six of the other seven connected components contain four vertices and three edges, but are not depth one trees. Each of these six components thus consists of a "central" n-value that has a factor in common with each of two other, co-prime n-values. The remaining connected component is the most intriguing – or suspicious – as it is a complete graph on nine vertices (K_9): nine primes, each of whose $\binom{9}{2} = 36$ pairwise products occurs as n-value.

Denoting the primes identified with the vertices of the graph by p_1, p_2, ... (using an ordering naturally implied by our representation), Figures 3.4, 3.5, and 3.6 depict the largest depth one tree, the six four-vertex components, and the K_9, respectively, with the edge labels indicating the number of X.509 certificates and PGP keys containing the corresponding n-value as RSA modulus. Note that all moduli in the K_9 occur quite frequently.

Any two n-values associated to edges in the same depth one tree can be factored. Two n-values associated to other edges can be factored if the edges are adjacent (i.e., share a vertex), or one finds a path connecting them. For non-adjacent edges in the same connected component from Figure 3.5 that is the unique central edge, for edges in the K_9 many paths are possible. All required edges are in our set of n-values.

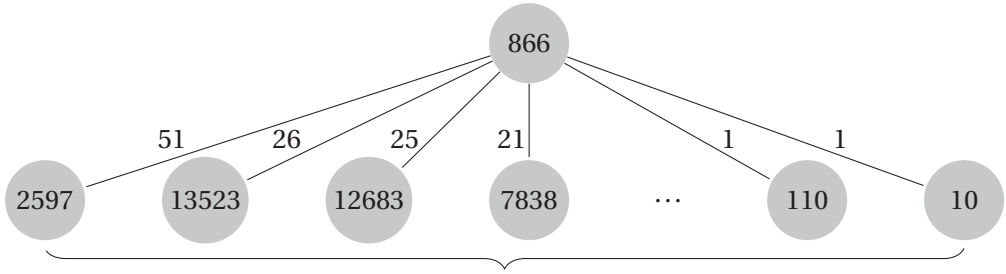


Figure 3.4 – The largest depth one tree found, on 4628 vertices, with the 512-bit prime p_{866} as root and leaves p_{2597}, p_{13523}, …, p_{10}. The edges correspond to 4627 distinct 1024-bit RSA moduli, with labels indicating the number of distinct X.509 certificates and PGP keys containing the RSA modulus corresponding to the edge, for a total of 5321 certificates. All certificates have expired and use SHA1 as hash function.

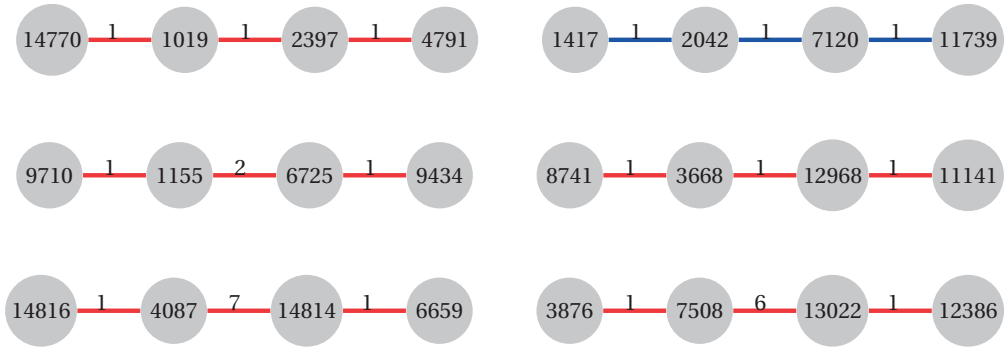


Figure 3.5 – Six connected components consisting of four vertices, with labels as in Figure 3.4. The eight primes in the top two components are 512 bits long, the other 16 are 256-bit primes). The red edges correspond to RSA moduli contained in certificates that will not expire anytime soon and that use SHA1 as hash function. The blue ones will expire soon and use MD5.

Affected RSA moduli and certificates. The 1995 components contain 14901 vertices and 12934 edges: 14901 distinct primes fully factoring 12934 distinct n-values (0.2% of the total), 11699 of which each occurs as RSA modulus in a single X.509 certificate or PGP key, and 1235 occurring more than once in, in total, 9720 certificates and keys. Thus, 11699 + 9720 = 21419 X.509 certificates and PGP keys are affected. Note that affected moduli are much more frequently shared than non-affected ones. None of the affected moduli are blacklisted.

Of the primes, 14592 are 512 bits long, 307 are 256-bit, and the remaining two have 257 bits. Of the n-values, 214 are 512 bits long, and there are 12720 of 1024 bits. Of the 512-bit n-values thus factored, 47 occur as RSA moduli in 188 X.509 certificates that have not expired and use SHA1. Of the factored 1024-bit n-values, 3201 occur as RSA moduli in 5250 certificates that have not expired and that use SHA1, of which 617 are regular non-self-signed end-user certificates with "`CA=false`" (with 390 distinct RSA moduli). The majority (4633, with 2811 moduli) has "`CA=true`" and is self-signed, of which 727 (304 moduli) have been used to certify

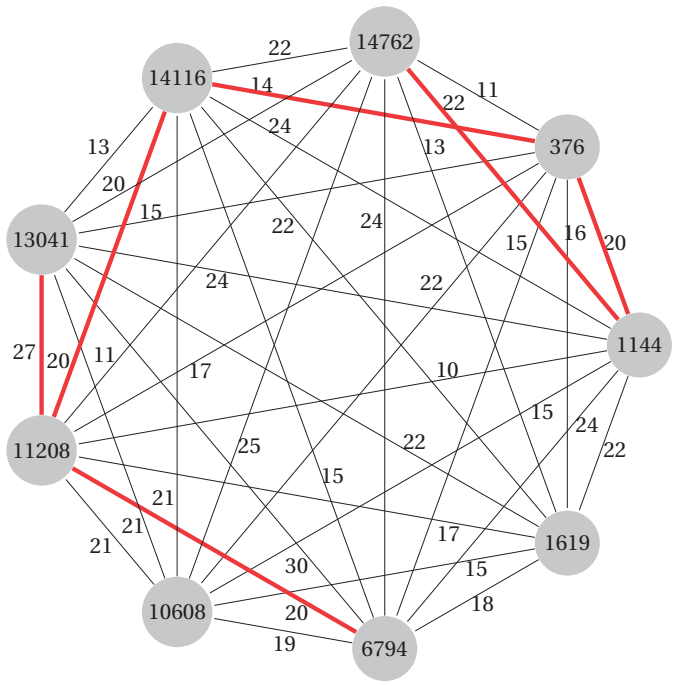


Figure 3.6 – Connected component consisting of nine vertices, corresponding to primes p_{376}, p_{1144}, …, p_{14762} (all 512-bit). With labels as in Figure 3.4, in total 687 X.509 certificates are involved. Six of those certificates have not expired yet, use SHA1 as hash function (as opposed to MD5), and have "`CA=false`"; the red edges correspond to the RSA moduli contained in those six certificates.

other RSA moduli (none among our collection of affected moduli). These 727 certificates share their "`issuer`"-field and the 304 RSA moduli occur in depth one trees not containing moduli owned by others. So, this security issue is reasonably contained. But 4445 of the 5250 certificates (and 537 of the 617 end-user ones) have no relation to that `issuer` and have a wide variety of "`issuer`" and "`subject`"-fields. This could be a security concern. We do not and will not reveal what types of users or devices are affected. We note, however, that our data give us more reason for concern than reported elsewhere (cf. [Hen12]) and that affected "flesh and blood" users that we talked to were not pleased[3].

Discussion. Generation of a regular RSA modulus consists of finding two random prime numbers. This must be done in such a way that these primes were not selected by anyone else before. The probability not to regenerate a prime is commensurate with the security level if NIST's recommendation [U.S09, page 53] is followed to use a random seed of bit-length twice the intended security level. Clearly, this recommendation is not always followed.

Irrespective of the way primes are selected (additive/sieving methods or methods using fresh random bits for each attempted prime selection), a variety of obvious scenarios is conceivable where poor initial seeding may lead to mishaps, with duplicate keys a consequence if no "fresh"

[3] "Donnerwetter!"

local entropy is used at all. If the latter is used, the outcome may be worse: for instance, a not-properly-random first choice may be prime right away (the probability that this happens is inversely proportional to the length of the prime, and thus non-negligible) and miss its chance to profit from local entropy to become unique. But local entropy may lead to a second prime that is unique, and thus a vulnerable modulus.

The above may, to some extent, explain the occurrence of duplicate RSA moduli and depth one trees. But we cannot explain the relative frequencies and appearance of these mishaps. Neither do we understand how connected components in Figures 3.5 and 3.6 may be expected to appear other than by very poor seeding or intentional malfeasance. Also, the great variation of issuers and subjects of the affected X.509 certificates (including the K_9) is disconcerting. No correlation between certification time and vulnerability of keys was detected. Vague, hand-waving arguments suggest that some of the devices involved may have used about 32 bits of entropy.

Avoiding two random choices during RSA modulus generation is straightforward (cf. [Len98]). But the resulting moduli may have other, as yet unpublished weaknesses (we are not aware of serious ones). It is better to make sure that cryptographic keys are generated only after proper initialization of the source of randomness.

3.4 ElGamal, DSA, and ECDSA

In this section we present the results of various counts and tests that we conducted on the data labeled as ElGamal, DSA, or ECDSA public keys. In neither collection did we find any of the numbers from [Moo08] (cf. Debian OpenSSL vulnerability [YRS+09]).

3.4.1 ElGamal

An ElGamal public key consists of a triple (p, g, y) where p is prime, g is a generator of the multiplicative group $(\mathbf{Z}/p\mathbf{Z})^*$ or a subgroup thereof of small index, and y is an element of $\langle g \rangle$. The secret key is an integer $x \in \{0, 1, \ldots, p-2\}$ with $g^x = y$.

Correct ElGamal keys. Among the PGP keys, 2 546 752 are labeled as ElGamal public keys. Three are incomplete and were discarded. Of the remaining triples 82 contain a composite p-value, resulting in 2 546 667 triples with correct p-values. Almost half (38) of the wrong p-values share a pattern with 65.6% of the p-values in the correct ElGamal keys, cf. below.

Restricting to the triples (p, g, y) with prime p-values, a triple is a correct ElGamal public key if $y \in \langle g \rangle$. To verify this the order of g, and thus the factorization of $p-1$, is needed. This is easy for *safe primes* (i.e., primes p for which $(p-1)/2$ is prime), but may be hard otherwise. The order of g could be established for 70.8% of the triples (65.6% with safe primes, 5.2% with primes p for which $(p-1)/(2m)$ is prime and $m > 1$ has only small factors) and could

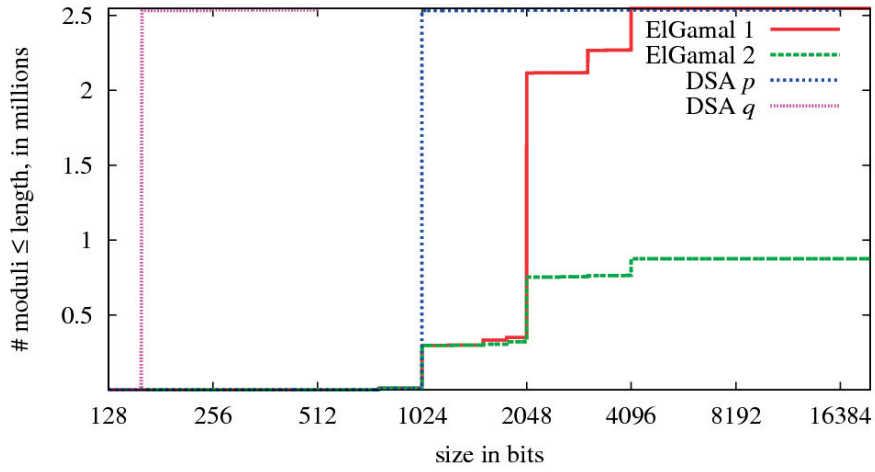


Figure 3.7 – Cumulative numbers of p and q-sizes in ElGamal and DSA public keys, indicated by "ElGamal 1", "DSA p", and "DSA q"; cumulative sizes of the distinct ElGamal p-values is indicated by "ElGamal 2".

reasonably be guessed for the other 29.2% (almost all with primes p for which $(p-1)/2$ is composite but has no small factors). For at least 16.4% of the ElGamal keys the g-values do not generate $(\mathbf{Z}/p\mathbf{Z})^*$. This led to 33 failed membership tests $y \in \langle g \rangle$, i.e., an insignificant 0.001% of the triples. Note that if $y \in \langle g \rangle$ a secret key exists; it does not follow that the owner knows it. A handful of triples were identified with peculiar y-values for which it is doubtful if a secret key is known to anyone.

Shared ElGamal keys. Six of the ElGamal keys occur twice: two keys with two unrelated owners each, and four keys occurring twice but with the same owner.

ElGamal key sizes. Figure 3.7 depicts the cumulative p-sizes in the set of 2 546 628 correct ElGamal keys. There are 1437 different p-sizes, ranging from thrice 256 bits to nine times 16384 and once 20000. Most frequent are 2048 bits (69.3%), 1024 (11.2%), 4096 (10.8%) and 3072 (5.8%) followed by 1536 (1.3%), 1792 (0.7%), 768 (0.4%), and 1025 (0.04%).

Shared primes, generators. Primes and generators may be shared. Among the 2 546 628 distinct ElGamal keys 876 202 distinct p-values (and distinct (p,g)-pairs) occur. Despite this high duplication rate, only 93 distinct p-values occur more than once. The four most frequent p-values are "similar". Let $p(x,L)$ denote the least safe prime $\geq x \bmod 2^L$. There is an integer v such that $p(v,L)$ for L-values 2048, 4096, 3072, 1536 occurs as p-value in 52.4%, 6.5%, 5.6%, and 1% of the ElGamal keys, respectively ($p(v,1024)$ occurs twice, $p(v+2^{510},512)$ once, and as noted above parts of v also occur in incorrect ElGamal keys). We suspect that these p-values, of different sizes, were generated using similar software and identical random seeding (if any), and from the least significant bit up to the most significant one.

All $p(.,L)$-values use $g=2$, which for $L=2048$ generates $(\mathbf{Z}/p\mathbf{Z})^*$, but for the others an index

two subgroup thereof. Overall, $g = 2$ occurs most frequently (70.8%), followed by $g = 5$ (19.5%), 6 (4.7%), 7 (1.9%), 11 (1.3%), and 13 (0.9%), with a total of 76 distinct g-values. No g-values were found that do not have a large order, but for at least 9.6% of the distinct (p, g)-pairs the g-values do not generate $(\mathbf{Z}/p\mathbf{Z})^*$. We can only give a lower bound because, as pointed out above, we failed to find *any* prime factor of 29.2% of the $(p-1)/2$-values in the ElGamal keys (which turns out to be 87.7% of the distinct $(p-1)/2$-values in the set of distinct p-values). Thus, we cannot be certain that the corresponding generators were properly chosen; consistent failure of all ECM factoring attempts of these numbers suggests, however, that they were well chosen.

Among the distinct ElGamal keys, all y-values are distinct, which is as expected because distinct (p, g)-pairs have negligible probability to lead to the same y-value (and identical (p, g)-pairs with identical y-values have already been identified and removed). The secret keys, however, may still be the same. But as there is no way to tell if that is the case (for distinct (p, g)-pairs) there is no serious security risk even if they are.

3.4.2 DSA

A DSA public key is a four-tuple (p, q, g, y) where p and q are primes with q dividing $p-1$, the element g generates an order q subgroup of the multiplicative group $(\mathbf{Z}/p\mathbf{Z})^*$, and y is an element of $\langle g \rangle$. The secret key is the integer $x \in \{0, 1, \ldots, q-1\}$ with $g^x = y$.

Correct DSA keys. Among the PGP keys and X.509 certificates, 2 536 959 and 141 four-tuples, respectively, are labeled as DSA keys. All four-tuples were first checked for correctness, casting them aside at the first test they failed. The tests were conducted in the following order: T1: primality of p; T2: primality of q; T3: divisibility of $p-1$ by q; T4: order of g equals q; and T5: order of y equals q. An insignificant 0.002% (66) of the PGP four-tuples are incorrect with failures 12×T1, 2×T2, 10×T4, and 42×T5 (where T2 failed twice for the same q-value, as it occurred twice). The X.509 DSA four-tuples passed all tests. Some of the failures may be due to transcription errors, as they occur in four-tuples that differ from correct ones in a few hexadecimal positions.

Shared DSA keys. The remaining 2 536 893 PGP DSA keys contain very few duplicates: one key occurs thrice (with possibly double ownership) and two keys occur twice each (each with single ownership), resulting in a total of 2 536 889 distinct PGP DSA keys. Although all 141 X.509 DSA keys are distinct, 95 of them are also among the PGP DSA keys, resulting in a total of $2\,536\,889 + 141 - 95 = 2\,536\,935$ DSA keys. We have not checked ownerships of these 95 duplicate DSA keys.

DSA key sizes. The cumulative p and q-sizes in the set of 2 536 935 DSA keys are depicted in Figure 3.7. There are nine different q-sizes: all except 0.2% (5012) are 160, with 256, 224, and 232 the most frequent exceptions occurring 4016, 702, and 249 times, respectively. The smallest and largest q-sizes are 160 and 512, the latter with seven occurrences. With 78

different sizes the variation among p-sizes is larger, though nowhere close to the variation among ElGamal p-sizes. All except 0.6% (15457) of the p-sizes are 1024, with 768, 2048, 3072 and 512 the most frequent exceptions with 9733, 3529, 1468 and 519 occurrences, respectively. The smallest and largest p-sizes are 512 and 16384, the latter with a single occurrence.

Shared primes, generators. Distinct DSA keys may contain identical p, q, or g values. In total 2 535 074 distinct p-values occur, with 2 535 037 distinct primes occurring once, 22 occurring twice, five occurring thrice, etc., up to a prime occurring 969 times (note the difference with ElGamal). Not surprisingly (but not necessarily, as the same q-value may give rise to many different p-values), the overall counts and numbers of occurrences are the same for the distinct q-values and the distinct (p,q)-pairs. Although the generator also allows considerable variation, the number of distinct (p,q,g)-triples is the same too. For all except 265 of the unique (p,q,g)-triples, the generator g equals $2^{(p-1)/q}$. We have not been able to determine how the other 265 generators were chosen.

The y-values are all distinct among the distinct DSA keys – given that shared keys were already removed, identical y-values would have been odd indeed. The same remark as above applies concerning identical secret keys.

3.4.3 ECDSA

The only interesting fact we can report about ECDSA is the surprisingly small number of certificates encountered that contain an ECDSA key (namely, just one), and the small number of certificates signed by ECDSA (one self-signed and a handful of RSA keys). As long as one subscribes to the notion of a standardized curve over a finite field of prime cardinality of a special form, as opposed to a randomly but properly chosen curve over a non-special prime field (cf. [LM10]), there is nothing wrong with the curve parameters `secp384r1`. It offers (in "theory") about 192 bits of security which makes it, security-wise, comparable to 8000-bit RSA moduli (n) and ElGamal or DSA finite field sizes (p), and 384-bit DSA subgroup sizes (q).

3.4.4 ElGamal and (EC)DSA.

Not surprisingly, the intersection of the sets of p-values for ElGamal and for DSA is empty. We have not tried hard to retrieve any of the secret exponents, i.e., (for ElGamal and DSA) x-values such that $g^x = y$, but have checked that none is less than 2^{12} in absolute value.

Random nonces in ElGamal and (EC)DSA. Unlike RSA, during signature generation ElGamal and (EC)DSA require a random nonce that should be entirely unpredictable (cf. [ElG85, NS02, NS03]). We are not aware of any studies that verify whether or not the nonces are properly chosen (with the notable exception of [Dar11]). Collecting data for such a study requires a much more intrusive type of data collection and may be considered unethical. Note, however, that a mishap in the form of a poorly chosen nonce affects *only* the party that makes the poor

choice, but does not affect any other party. In particular the choice of two identical nonces for distinct ElGamal or DSA parameters does not affect anyone but the two users involved.

Discussion. Both for ElGamal and DSA a small number of keys were identified that are shared among unrelated parties. This may be a security concern. Furthermore, there were some ill-formatted keys that cannot be expected to work and that should be of insignificant security concern. From the point of view of this work, the main security concern for ElGamal and (EC)DSA is the generation of the random nonce; this is a key usage but not a key generation issue and therefore beyond the scope of this work.

3.5 Conclusion

We checked the computational properties of millions of public keys that we collected on the web. The majority does not seem to suffer from obvious weaknesses and can be expected to provide the expected level of security. We found that on the order of 0.003% of public keys is incorrect, which does not seem to be unacceptable. We were surprised, however, by the extent to which public keys are shared among unrelated parties. For ElGamal and DSA sharing is rare, but for RSA the frequency of sharing may be a cause for concern. What surprised us most is that many thousands of 1024-bit RSA moduli, including thousands that are contained in still-valid X.509 certificates, offer no security at all. This may indicate that proper seeding of random number generators is still a problematic issue.

The lack of sophistication of our methods and findings make it hard for us to believe that what we have presented is new, in particular to agencies and parties that are known for their curiosity in such matters. It may shed new light on NIST's 1991 decision to adopt DSA as digital signature standard as opposed to RSA, back then a "public controversy" (cf. [DLL+93]); but note the well-known nonce-randomness concerns for ElGamal and (EC)DSA (cf. Section 3.4.4) and what happens if the nonce is not properly used (cf. [Dar11]).

Factoring one 1024-bit RSA modulus would be historic. Factoring 12720 such moduli is a statistic. The former is still out of reach for the academic community (but anticipated). The latter comes as an unwelcome warning that underscores the difficulty of key generation in the real world.

Acknowledgements

We thank Peter Eckersley from EFF for his invaluable assistance and Don Johnson for pointing out [Joh99] to us. We gratefully acknowledge key collection and other efforts by Benne de Weger, Philippe Joye, Paul Leyland, Yann Schoenberger, Christoph Schuba, Deian Stefan, Fabien Willemin, Maryam Zargari and others that need to remain anonymous. The first author appreciates the legal advice from Mrs. Chardonnens. James P. Hughes participated in this project in his individual capacity without sponsorship of his employer or any other party.

This work was supported by the Swiss National Science Foundation under grant number 200020-132160.

4 Privacy Leaks through File Sizes

4.1 Motivation

Encryption of stored data is an unavoidable security requirement when one wishes to outsource storage infrastructure to a third party. Low overhead from the cryptographic layer is obviously a desirable property.

Stream ciphers produce ciphertexts the same size as their corresponding cleartexts, down to the single bit if needed. This can also be achieved with block ciphers by using the appropriate modes:

- Counter (CTR) mode produces a cryptographically secure pseudorandom stream by concatenating the ciphertexts produced by encrypting an increasing counter initialized at a secure IV.
- Output Feedback (OFB) mode likewise produces a stream by iterating the encryption function over a secure IV, and concatenating the ciphertexts of each iterated step.
- Cipher Feedback (CFB) mode produces a stream by encrypting the stream ciphertext itself, usually with a fixed offset of one block. It is thus not identifiable to a stream cipher, but still maintains the property that the ciphertext can be cut down to the exact size of the plaintext without losing information.

If we suppose that the keys and initialization vectors are stored elsewhere, as part of the metadata, such setups have no storage overhead. While it may seem desirable in terms of space efficiency, it also allows a passive attacker to observe the exact size of the cleartext. It is also still the case if the system has a fixed or predictable overhead, such as when it is just storing a single initialization vector per object.

While this information leak may be less of a concern in a context where message size conveys little information, one can wonder if this is still the case for immutable systems archiving large

files. In this chapter, we will try to assess the potential security consequences of this property on realistic systems.

For the following discussion, we will follow the IEC standard [IEC08]: Classical SI prefixes used with bytes (kB, MB, GB…) refer to their habitual powers of 1000, while binary prefixes (kiB, MiB, GiB…) refer to powers of 1024.

We can compute a coarse estimate of the probability of a file to have a unique size. Let us take movies as an example. At the time of this writing, IMDB statistics [IMD] advertise knowledge of 340k movies. As the worst case, consider that all of them are available in storage, with inefficient duplication of content for all the possible audio languages, subtitle languages, and codec choices; a conservative estimation is 10^4 for an upper bound on the average number of variants per movie (100 audio languages, 10 subtitle language choices for each audio variant, and 10 different codecs). Let us further assume that file sizes are uniformly distributed in a range of the order of 2×10^9 bytes (between 1 and 3 GiB).

Assume we have k distinct files being randomly mapped over a range of n possible distinct file sizes. We are interested to compute the probability P that some observed file size is unique. Let P_0 be the probability that a file size is mapped by no files, P_1 be the probability that a size is mapped by exactly one file, and P_{2+} the probability that a size is mapped by two files or more. Deriving P_0 and P_1 from a binomial law, we have:

$$
\begin{aligned}
P_0 &= \left(\frac{n-1}{n}\right)^k \\
P_1 &= \frac{k}{n}\left(\frac{n-1}{n}\right)^{k-1} \\
P_{2+} &= 1 - P_0 - P_1 \\
P &= \frac{P_1}{P_1 + P_{2+}} = \frac{P_1}{1 - P_0}
\end{aligned}
$$

For the chosen $n = 2\times10^9$ and $k = 340\times10^7$, we compute a probability $P \approx 0.38$ of being mapped to a unique file. While this is a very coarse estimation, it shows that we cannot simply dismiss the possibility that the exact size of a large multimedia file conveys significant information about its contents.

In systems like Tahoe [WOW08], the problem is exacerbated by the use of convergent encryption, a mechanism which allows deduplication of encrypted data. A convergent encryption scheme ensures that the same contents, encrypted by two different users, will produce the same ciphertext (hence the ciphertexts are said to converge). One can build a convergent encryption scheme by requiring that keys and initialization vectors are deterministically derived from the actual file contents, for instance by a cryptographically secure hash function.

Convergent encryption has the drawback that anyone can probe the storage system and tell if some content already exists. In extreme cases such as very small files with mostly known content, it might even be possible to guess the full contents through brute force, as seen in [HPSP10].

To avert these attacks, the convergent key derivation process might be salted by a *convergence secret*. Only users sharing the same convergence secret will benefit from mutual deduplication of their storage space, at the cost of potentially exposing themselves to these side-channel leaks. A user who does not care for such attacks may use a publicly agreed-on value, to benefit from maximal deduplication, while a paranoid user may use a random secret value, and get no deduplication at all.

In order for the convergent scheme to work, all participating users must obviously agree on ciphers, and key derivation algorithms. In the case of Tahoe, for instance, using AES in CTR mode is part of the agreement.

4.2 Typical Block Sizes

For practical reasons, when operating close to the hardware layer, storage systems will typically operate on blocks for efficiency reasons. The optimal block size poses a trade-off between wasted space, and compactness and efficiency in the metadata management. In fact, the ubiquitous use of the byte as the smallest storage unit is already a choice of block size, albeit a tiny one. The native size for block storage on Unix is 512 bytes. Ext4, the current default filesystem for Linux, uses blocks between 1 and 64kB, the default being 4kB. AES operates on blocks of 16 bytes. BitTorrent uses two levels of granularity: *pieces*, offering granularity for the integrity checks (of variable size, typically of a power of 2, between 2^{18} and 2^{24} bytes), and *blocks*, offering granularity for the network transfers, for which the *de facto* universal standard is 16kB [Coh08].

4.3 Data Collection

We chose to study the set of files publicly shared over BitTorrent. In addition to being a well alive and popular technology [PGES05], file names and sizes can be extracted from the metainfo blocks of `.torrent` files. Data is cryptographically authenticated, and `.torrent` files can in turn be obtained from community-vetted sources. The file names extracted from these are presumably more reliable than with other P2P filesharing technologies, where the cost of registering and disseminating a bogus entry can be very low.

What we discovered empirically deviates from models used to represent file size distribution in typical computer systems [Mit04]. Nevertheless, while not representative of the files a single user might store on a home computer, content sampled from P2P networks is arguably more representative of typical content which would be most likely to be found in cloud-based public

storage, and benefit the most from deduplication.

We scraped public repositories such as The Pirate Bay [SNS], as well as various other, private, sources. We acquired around 7M `.torrent` files, containing metadata for a total of 12.4M downloadable files. The downloadable data represents a total volume of 437.8TB (398.2 TiB).

4.3.1 Architecture of BitTorrent

The unit of sharing in BitTorrent is not the file, but the torrent. A torrent is an immutable set of related files (possibly a singleton set), cryptographically authenticated. The metadata for the torrent are contained in a small `.torrent` file. These files initially needed to be distributed through an external channel.

Most datastructures in BitTorrent uses Bencoding [Coh08], a serialization format supporting the usual dynamic datastructure paradigm (integers, strings, lists, and associative lists). A `.torrent` file is itself a Bencoding representation of a *metainfo* datastructure. The largest part of the metainfo structure is usually the *info* datastructure, which contains the unchanging information pertaining to the downloadable file set, such as the file names, file sizes, and the chosen piece size. In short, the info structure contains information about *what* to download, and the rest of the metainfo structure contains information about *how* to download it. Thus, information outside the info structure may change at any time without breaking the authentication of the data.

To perform the cryptographical integrity check, the client considers the concatenation of all the files in the torrent, arranged by lexicographic order of their respective file names, and slices this stream in pieces of regular size[1]. The SHA-1 of every piece is computed and stored in the *info* structure. BitTorrent clients may negotiate transfer of multiple pieces in parallel, and use the hashes to verify the integrity of the downloaded data while the download is in progress. They may safely start uploading verified pieces, and can legitimately punish[2] peers who transmit incorrect data, without losing efficiency.

Because Bencoding imposes a canonical ordering when serializing associative arrays, the info structure has a deterministic, consistent serialization, that can then itself be authenticated by its SHA-1[3] hash, which is called the *infohash* value.

[1] The block size is uniform across a single torrent, but can otherwise be freely chosen by the torrent publisher. It offers a trade-off between the metadata size and the integrity check granularity. The last piece may be smaller than the chosen piece size.

[2] Typically by uploading to faulty peers at a reduced rate, or refusing to upload anything at all.

[3] There are no plans to upgrade to a more secure hash function. However in this case, collisions are not much of a problem; second preimage attacks would be.

4.3.2 Trackerless operation

The original operation mode of BitTorrent relies on a *tracker* service, a compromise between a fully centralized index service (like the original Napster design, which proved very fragile under pressure of censorship) and completely distributed search mechanisms like Gnutella (based on broadcasting, slow and wasteful) or Kademlia [MM02] (a Distributed Hash Table based on the XOR metric, reasonably efficient, but less suitable for operation on private networks, and without accounting support). The tracker service acts as an index of all the peers interested in downloading or sharing the data; peers announce themselves periodically[4], and receive a list of other active peers. Peers can contact each other at any time and negotiate a data transfer. However, instead of having a single global tracker service, each torrent can run on a separate tracker (the address of the tracker(s) is part of the `.torrent` data. This allows BitTorrent to operate as a data distribution protocol on private networks, and not only for public filesharing.)

In this original design, the tracker service is a target of choice for censorship. If a tracker is only responsible for a small set of torrents, disabling it will cripple distribution of this torrent set, and nothing else.

As a consequence of the cat-and-mouse game between filesharers and censorship attempts, various improvements have been developed to make the protocol more robust against censorship. The three more important mechanisms are:

- A *Distributed Hash Table* complements or replaces the original tracker mechanism. The DHT service stores arbitrary key-value pairs in a fully distributed fashion, and cannot be easily shut down. Peers register themselves by adding their own addresses under the key for the infohash they are interested in, and query the same key to find out about other peers.

- The *Peer Exchange Extension* allows peers to exchange contacts directly, making the role of the tracker less critical. In addition, cautious peers may choose not to announce themselves to a tracker, and only query the DHT without registering in it, to become less visible to outsiders (they will only reveal themselves, through peer exchange, to peers actively engaging in the download, or at least faking it up to a certain point; they cannot be spotted just by hitting the tracker or querying the DHT)

- The *Metadata Extension* allows peers to request and exchange the full content of the metadata structure for the torrent.

It should be noted that these extensions were developed in response to rather arbitrary jurisprudence. Initially, large sites like The Pirate Bay [SNS], hosting torrents containing legally

[4] Typically every 5 minutes.

dubious contents, did hide behind the fact that they were only hosting metadata, and not actual contents. When this position became unreasonably hard to maintain, they removed the actual `.torrent` files, instead offering only an index of interesting infohash values, in the form of Magnet Uniform Resource Identifiers. Magnet URIs are but one form of URIs [Cha01, CSF+08] that contain nothing but a cryptographic hash of the target data (in our case, the infohash). For some reason, metadata are considered less problematic when passed through a cryptographic hash function, and reduced from a few kilobytes to a hundred of bytes, even when the cryptographic hash function can be easily inverted thanks to the availability of the peer-to-peer network.

When the three extensions above are implemented, the infohash is the only piece of information required to acquire the torrent contents in an authenticated way. The process is as follows:

- If necessary, bootstrap the connection to the DHT, through the use of well-known nodes.[5]
- Query the DHT with the known infohash as the key, obtain a list of peer addresses.
- Contact some peers, announce the infohash value, and identify peers supporting the Metadata Extension.
- Make a metadata request, obtain the full info structure.
- Use the Peer Exchange Extension to obtain more peer addresses from the existing ones.
- Proceed normally.

4.3.3 Fast Resolution of Magnet URIs

Because we did not want to exclude sites offering only magnet links for our study, we had to acquire a way of retrieving the `.torrent` files associated to a particular magnet link. At the time of this investigation, no easy publicly available, scriptable tool existed to perform this task, short of running a full BitTorrent client and actually running the download.

The probable reason is that performing this operation already requires implementing the most complex components of a full BitTorrent client: a DHT client, a Bencoding serializer and deserializer (because messages for the metadata request use it), and network code to contact the peers. Once a tool reaches this stage of development, there is little work remaining to turn it into a complete BitTorrent client.

We used code from Etorrent [And], a BitTorrent client written in Erlang, and adapted it to perform the minimal steps required to resolve a magnet link. In particular, the program joins

[5] This only needs to be done once per client, and only on the first run or after a long period of inactivity.

the DHT, and is able to query it, but never registers itself under any infohash value. It does connect to peers, and makes a metadata transfer request, then disconnects immediately. All operations are cheap to do asynchronously thanks to Erlang, allowing us to resolve our large collection of magnet links much faster than a naive implementation.

Unfortunately, because we have to disclose the infohash value we are interested in, we cannot ensure that the peer will not associate our address with the infohash and propagate it further, at least temporarily, through the Peer Exchange extension. Testing this may become the subject of future work.

While performing this process, we observed that a very significant fraction (more than 25%) of the peers registered in the DHT refused to answer our connection requests. This could be related to network filtering issues[6], but another possible explanation is the parasitic use of the DHT to run amplified Distributed Denial-of-Service attacks. This kind of behaviour has already been investigated in [EDGM07].

4.3.4 Performance

We found operating on a collection of millions of very small files to be quite taxing for the default Linux filesystem we used (ext3). When random access to a specific torrent is not required, it is at least an order of magnitude faster to store the collection packed in a single tar file, and to unpack it on-the-fly and stream the torrent files to the analysis process. The sample code we used to do so is available on Github[7].

4.4 Analysis and Results

Because fetching actual data would be unreasonably costly in terms of bandwidth and storage, we restrict ourselves to analysing the metadata only. Unfortunately, the hashes stored in the metainfo correspond to blocks not necessarily aligned with file boundaries.[8]. Therefore it is not in general possible to tell if two files of identical size are likely to be identical in content, unless the two torrents happen to use the same block size, and the respective offsets in the two torrent streams differ by exactly a multiple of this block size.

We thus restrict ourselves to using file names as identifiers for the content. A file name is not always a reliable predictor for content: typically, in the case of uncompressed ebooks, it is common for files to be named after the number of the page, without additional hints identifying the ebook itself. It is also relatively common for some archive formats to split a large archive across several fixed-size files, and give these files a name derived from a common regular pattern. Nevertheless, the relationship between file sizes and names can still be

[6] Operation behind Network Address Translation (NAT) gateways is widely known to be problematic.

[7] https://github.com/maugier/tortar

[8] Some clients address this problem by adding special zero-filled padding files, exactly of the size required to make the beginning of the next real file align with a block boundary, but this practice is not generalized.

Size (GiB)	Name
222.9	LIBRARY .zip
166.4	DJANDYW.TK ULTIMATE SAMPLES COLLETION.zip
120.9	Harry Potter.mpeg
116.6	Hetero.rar
86.6	AGI.rar
82.9	Ponibooru-All-Safe.zip
68.0	Robert.Gustafsson.Ultimate.Collection.SWEDiSH.BOX.PAL.DVDr.rar
63.7	Desktop.rar
56.4	Music, Pictures, and Meth.zip
55.6	Крепкий орешек.mkv

Table 4.1 – Largest files observed

objectively measured and used to predict file names.

We observed 6.35M distinct filename-filesize pairs (6.33M after removal of special empty padding files.) About 4k files had a size of zero. Of these empty files, 1.2k seemed to follow a systematic naming convention containing movie or TV show metadata (name, year and episode). It is thus an ad-hoc method for storing relevant metadata in the downloaded data itself, in a reliable cross-platform way.

The rest of the empty files have benign names. They are most likely the result of human error, the respective torrents having been created while the download from the original (non BitTorrent) source was not yet complete.

As an indication, the 10 largest files observed in the collection are in Table 4.1[9]. Interpretation of the file names is left to the reader.

4.4.1 File types

Table 4.2 shows the 20 most frequent file extensions, as announced by the metadata. In principle, we cannot check the accuracy of the extensions against the actual content, as we do not have access to it. However, as said before, the torrents and magnet links were obtained from sources which perform a community vetting process on them, so we expect the majority of them to at least not be deliberate fraud attempts.

The predominance of image files was somewhat surprising to us, but can be partially explained by the way ebooks are sometimes packaged (as uncompressed directories of one image per page, typically.)

Table 4.3 shows the 20 file extensions representing the largest data volume in the collection. Without surprise, video files are by far the largest space consumer.

[9] Errors in the file names are authentic.

Extension	Filenames	Extension	Filenames
.jpg	2701999	.wav	53350
.mp3	1239286	.txt	51929
.avi	229170	.cbr	43171
.flac	146078	.zip	40609
.png	134382	.srt	37573
.mp4	121384	.vob	34489
.mkv	106705	.wmv	30351
.pdf	105190	.exe	29437
.rar	92016	.flv	26996
.nfo	55560	.m4a	25976

Table 4.2 – Distinct file names for most frequent extensions

Extension	Volume (TB)	Extension	Volume (TB)
.mkv	117.752	.zip	4.762
.avi	109.163	.bin	4.599
.mp4	44.829	.mpg	3.098
.iso	41.902	.m4v	2.899
.rar	16.977	.ts	2.723
.m2ts	13.360	.exe	2.554
.mp3	11.387	.flv	1.906
.wmv	10.899	.jpg	1.630
.vob	8.743	.mov	1.395
.flac	5.492	.pdf	1.351

Table 4.3 – Data volume by advertised extension

To get a finer analysis, we group these extensions by media type, according to Table 4.4, and look at them as distinct collections.

Figure 4.1 shows the overall histogram of the number of different file names observed for some given size. Points at the bottom ($y = 1$) denote unique file sizes for which it is possible to unambiguously guess the filename (assuming prior knowledge that the file is part of this torrent collection). One can see that the first unique file sizes appear around 8kiB (below this bound, no accurate guesses can be made) and the last non-unique file sizes cluster around 17GiB (above this bound, file sizes are always unique).

Interestingly, if one presupposes knowledge of the media type (audio, video, ...), the respective histograms offer very different shapes, as seen in Figure 4.2:

- For audio, one expects the sizes to cluster around the approximate sizes, for typical encoding parameters, of a song of typical size.
- For images, the same clustering can be observed around typical wallpaper resolutions and their sub-multiples.

Media type	Extension	Notes
archive	.zip	Common under Windows
	.rar	For historical reasons, popular among the filesharing crowd
	.r[0-9][0-9]	RAR uses numbered extensions (.r00, .r01, ...) for split archives
audio	.flac	Free Lossless Audio Codec, popular among audiophiles
	.mp3	Historical *de facto* standard of filesharers
	.m4a	Container for Advanced Audio Coding (AAC)
	.ogg	Open multimedia container, but audio most of the time
	.wav	Historical PCM container
	.wma	Windows Media Audio
image	.bmp	Windows bitmap format
	.gif	Graphics Interchange Format
	.jpg	JPEG
	.png	Portable Network Graphics
video	.avi	Audio Video Interleave, historical Microsoft multimedia container
	.flv	Flash Video (Macromedia/Adobe)
	.mkv	Matroska, an open multimedia container
	.mp4	MPEG-4
	.m2ts	Native container format for BluRay discs
	.ts	MPEG Transport Stream, possibly from digital TV or satellite broadcasts
	.vob	Video Object, native format for DVD discs
	.wmv	Windows Media Video
iso	.iso	ISO 9660 format, "mirror images" of optical media (CD, DVD, BluRay...)

Table 4.4 – Media type by extension

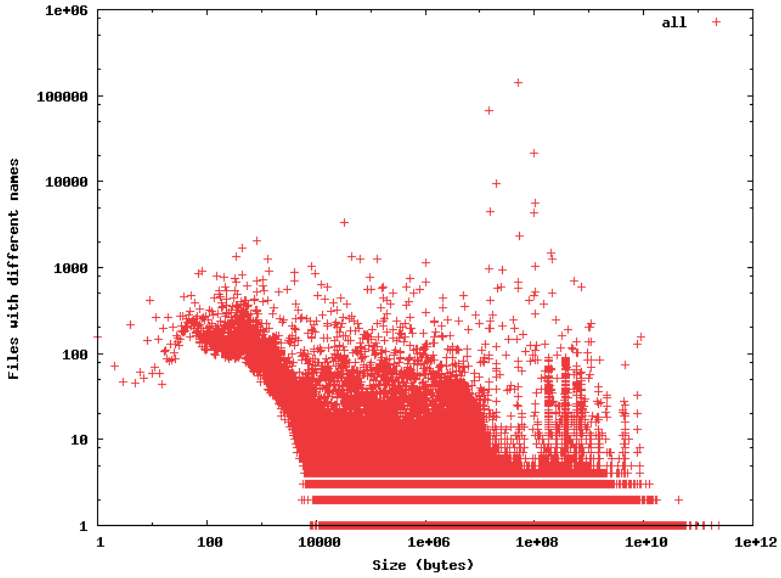


Figure 4.1 – Distinct file names for each possible file size, all types together

- For videos, we observe much tighter clusters around 367MiB, 576Mib, and 1.07GiB. We are not certain of the origin of the first. The second corresponds to the space available to the user on a 700MiB CD-R after it has been formatted with Microsoft's Live File System, their native packet writing format for Windows. The third corresponds to a limitation in the UDF disk format, which uses a 30-bit integer to store file sizes.

- For ISOs the strongest cluster occurs around 4.7GB, as expected for single-layer DVD images, the second strongest around 8.7GB (double-layer single-sided DVDs).

- Finally, compared to other categories, Archives have the largest amount of identical sizes. This is due in great part to split archives, historical formats consisting of several identically-sized blocks. While the original use for such splits was to easily split large datasets across removable media, it is still popular in some filesharing circles, as a low-tech alternative that allows parallel downloading across multiple endpoints (and races between uploaders) on top of old protocols, like FTP, that were not designed to support multihomed transfers at the file level.

Figure 4.3 shows, for each media type, the rank of every observed distinct pair of file name and size, when sorted by descending file size. Vertical drops are large sets of files sharing the same size. The flat segments at the start of each curve are rare, abnormally small files, that may indicate incomplete transfers.

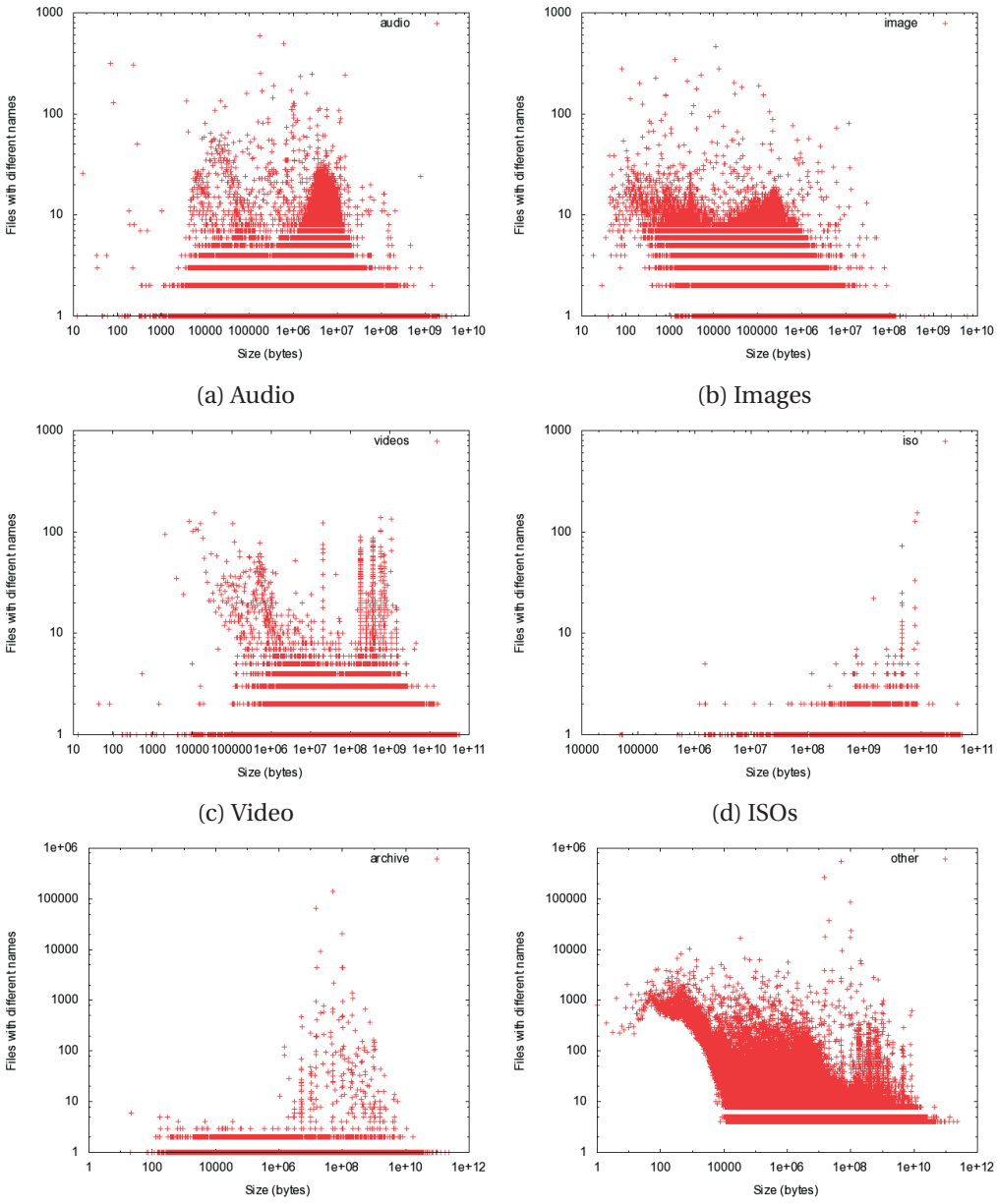
Figure 4.4 shows the cumulative probability distribution of uncertainty over the file name, given a fixed file size randomly chosen uniformly among all the possible observable file sizes. The curve gives, for a given entropy value (the logarithm base 2 of the number of possible file names), the amount of different file sizes having less than the specified uncertainty.

4.4.2 Entropy losses

To quantify the privacy loss caused by the disclosure of a file size, a good metric would be the average difference of entropy in the distributions of file names with and without some knowledge of the size. This knowledge may be of the exact byte-wise file size, or it may be block-wise for various typical block sizes.

For the rest of this section, we consider a random uniform choice over all possible distinct file entries. Entries are distinct if they differ by size, or by filename (without the base path), or both.

Disregarding the file size information, the effective entropy of the file name distribution is 21.918 bits. It would have reached 22.594 bits if all 6.33M entries had distinct file names, but there is an ample supply of files with different sizes and identical names. The worst offenders are movie cover images (`cover.jpg`, `front.jpg`, `back.jpg`), VOB files ripped with their original internal-use-only names, executable installers (`setup.exe`), and files named



(a) Audio

(b) Images

(c) Video

(d) ISOs

(e) Archives

(f) Other

Figure 4.2 – Distinct file names for each possible file size, arranged by media type

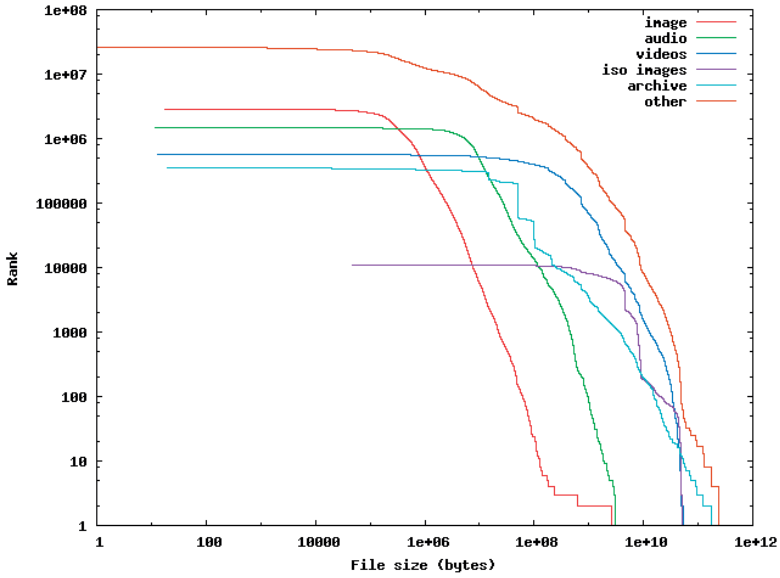


Figure 4.3 – Ranking by size of distinct files

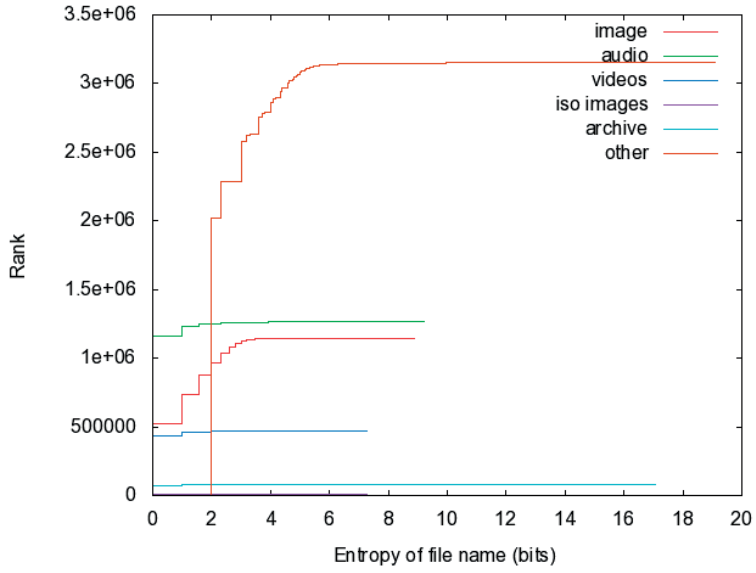


Figure 4.4 – Ranks of file name entropy, file size known up to the byte

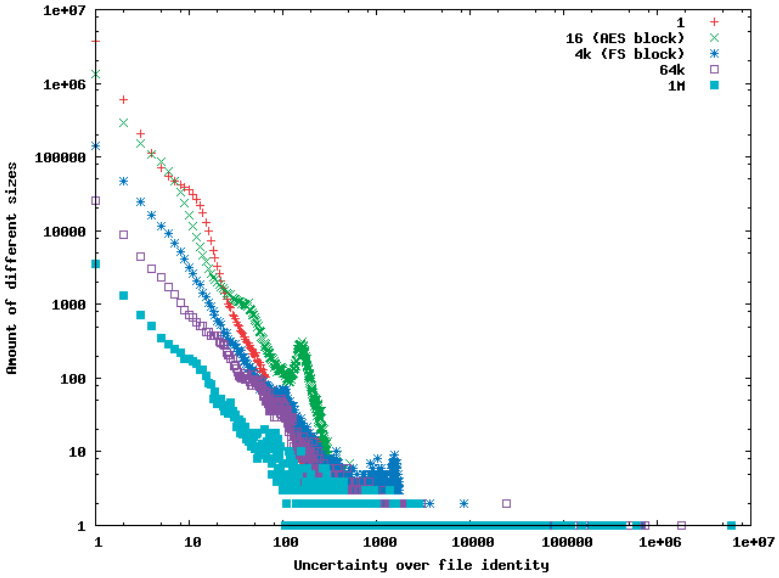


Figure 4.5 – Distinguishable file sizes, for given uncertainty over file name, with different block rounding

after just a single number (unpacked ebooks with one file per page, named `001.jpg`, `002.jpg`, ..., and music CDs with `Track 1.mp3`, `Track 2.mp3`, ...)

Figure 4.5 shows the distribution of entropy over the file name, given a randomly picked file size, when the sizes have been rounded up to some typical block size values. As expected, the curves are mostly decreasing, reflecting the increasingly lower probability to collide with another file size by chance. We do not know how to explain the irregularities in the curves, in particular the spike exhibited by 16-bytes blocks.

We then compute information leaked, on average, by observing the file size rounded up to a chosen block size. To do so, we take our collection of known file sizes, perform the rounding, then compute the entropy of the resulting distribution. Figure 4.6 shows the information leaked, on average, when block sizes are powers of 4, starting from 16 to 4^{16}.

We can see the privacy increases almost linearly with the block size, for a domain of realistic values ranging up to 1MB. After that, the benefits decrease, and the curve is of course expected to hit exactly zero when the block size becomes larger than the largest file in the collection.

Figure 4.7 shows the trade-off between information leaked, and total wasted space. The total wasted space is the sum, for every known file, of the difference between their actual size and their rounded-up size.

The remarkable similarity between the two figures is explained by the fact that wasted space increases almost linearly with block size.

Figure 4.7 also shows the trade-off for an adaptive padding scheme where file sizes are rounded

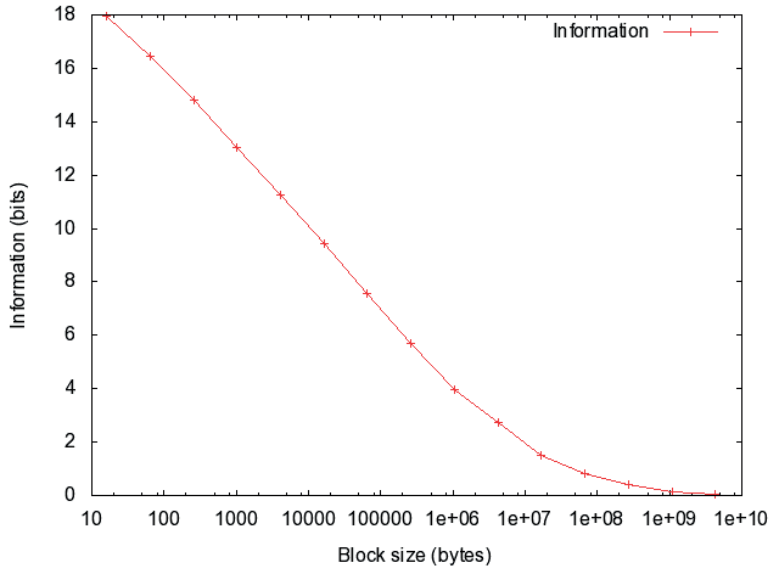


Figure 4.6 – Information in truncated file sizes, per block size

up the nearest larger power of respectively $\sqrt{2}$, 2, 4, and 8. Larger bases are more wasteful, and leak less information. Much to our disappointment, this scheme seems to offer a worse trade-off than a simple fixed block size. This may be explained by the fact that adaptive padding, as we tested it, is very wasteful for the largest files, but leaks less information about these large files, which are on average less common than the small ones.

4.5 Conclusion

Information leakage through file sizes, be they exact byte-wise size or up to rounding to a small block size, appears to be a legitimate concern in the case of media files, according to what we could measure from publicly available information. The obvious solution to this problem is to use an appropriate padding mechanism to make relatively close file sizes indistinguishable, enough for the uncertainty to go back to acceptable levels.

Padding will be wasteful, unless the padding scheme chosen makes the storage system aware of the padding. We can distinguish between two scenarios: padding the data transfers only, which will protect against a passive attacker listening on the communication channel, and will cost bandwidth, but not storage space; and padding the data, on top of the encryption layer (of which the storage system must not be aware), which protects against an honest-but-curious storage system, but costs storage space.

There will be a trade-off between wasted resources and privacy loss. In order to find the optimal one, one must first assign a cost to privacy loss, in order to compare it to storage costs. However, no matter which trade-off is chosen, if one assumes that privacy should be equivalent for most files, no matter their sizes, the size distribution shows that the padded

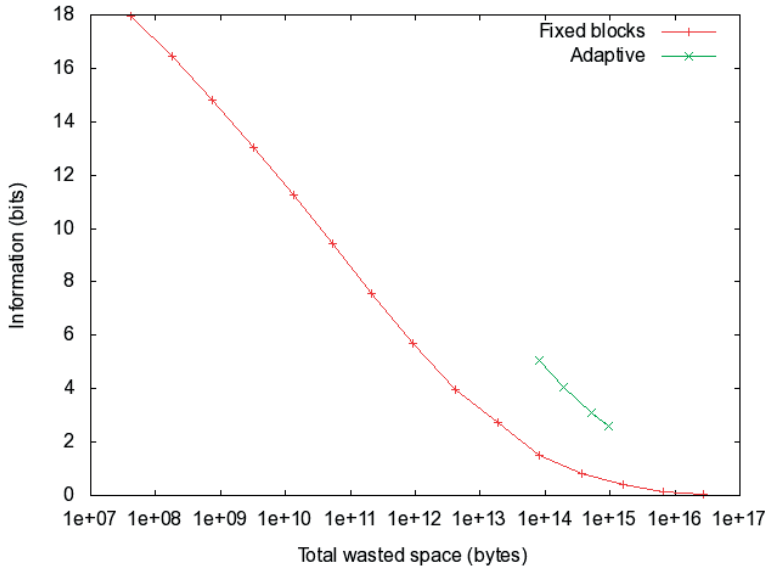


Figure 4.7 – Information in truncated file sizes, per wasted space

block size should be roughly proportional to the file size. Unfortunately, a naive padding scheme, for instance to the next power of two, or to the next fractional power of two (in effect, the next power of some integer root of two) seems to be strictly worse than a fixed block size. Variable-length blocks delimited by rolling hashes, as described in Section 1.1.3, may offer best results. Unfortunately, testing this hypothesis would require access to the actual data, not just the metadata we currently have.

Bibliography

[17912] One-way indexing for plausible deniability in censorship resistant storage. In *2nd USENIX Workshop on Free and Open Communications on the Internet*, Berkeley, CA, 2012. USENIX.

[ADDV15] Giuseppe Ateniese, Özgür Dagdelen, Ivan Damgård, and Daniele Venturi. Entangled encodings and data entanglement. In Feng Bao, Steven Miller, Sherman S. M. Chow, and Danfeng Yao, editors, *Proceedings of the 3rd International Workshop on Security in Cloud Computing, SCC@ASIACCS '15, Singapore, Republic of Singapore, April 14, 2015*, pages 3–12. ACM, 2015.

[AFYZ07] James Aspnes, Joan Feigenbaum, Aleksandr Yampolskiy, and Sheng Zhong. Towards a theory of data entanglement. *Theoretical Computer Science*, 389(1–2), December 2007.

[And] Jesper Louis Andersen. Etorrent, a bittorrent client written in erlang. https://github.com/jlouis/etorrent.

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

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

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

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

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

[CDL+00] Stefania Cavallar, Bruce Dodson, Arjen K. Lenstra, Walter M. Lioen, Peter L. Montgomery, Brian Murphy, Herman te Riele, Karen Aardal, Jeff Gilchrist, Gérard Guillerm, Paul C. Leyland, Joël Marchand, François Morain, Alec Muffett, Chris Putnam, Craig Putnam, and Paul Zimmermann. Factorization of a 512-bit RSA modulus. In Bart Preneel, editor, *Eurocrypt 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Heidelberg, 2000.

[Cer00] Certicom Research. Standards for efficient cryptography 2: Recommended elliptic curve domain parameters. Standard SEC2, Certicom, 2000.

[CGF10] Henry Corrigan-Gibbs and Bryan Ford. Dissent: Accountable group anonymity. In *17th ACM Conference on Computer and Communications Security (CCS 2010)*, number EPFL-CONF-212686, 2010.

[Cha82] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology, Proc. of Crypto 82*, 1982.

[Cha01] Justin Chapweske. Http extensions for a content-addressable web, 2001.

[Coh08] Bram Cohen. Bep-3: The bittorrent protocol specification, 2008.

[Cop93] Don Coppersmith. Modifications to the number field sieve. *Journal of Cryptology*, 6(3):169–180, 1993.

[CSF+08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, 2008.

[CSWH01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, pages 46–66. Springer-Verlag New York, Inc., 2001.

[Dar11] Darkmirage. PS3 completely cracked, 2011. http://www.darkmirage.com/2011/01/06/ps3-completely-cracked/.

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

[DLL+93] Yvo Desmedt, Peter Landrock, Arjen K. Lenstra, Kevin S. McCurley, Andrew M. Odlyzko, Rainer A. Rueppel, and Miles E. Smid. The Eurocrypt '92 controversial issue: Trapdoor primes and moduli (panel). In Rainer A. Rueppel, editor, *Eurocrypt '92*, volume 658 of *Lecture Notes in Computer Science*, pages 194–199. Springer, 1993.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[EDGM07] Karim El Defrawy, Minas Gjoka, and Athina Markopoulou. Bottorrent: misusing bittorrent to launch ddos attacks. In *Proceedings of the 3rd USENIX workshop on Steps to reducing unwanted traffic on the internet*, pages 1–6. USENIX Association, 2007.

[EFRS90] Paul Erdös, Ralph J. Faudree, Cecil C. Rousseau, and Richard H. Schelp. Subgraphs of minimal degree k. *Discrete Mathematics*, 85(1):53–58, 1990.

[Ele10] Electronic Frontier Foundation. EFF SSL Observatory. https://www.eff.org/observatory, 2010.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Blakley and David Chaum, editors, *Crypto 1984*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, Heidelberg, 1985.

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

[Fre11] Free Software Foundation, Inc. *GMP: The GNU Multiple Precision Arithmetic Library*, 2011. Available at http://www.gmplib.org/.

[G+09] Craig Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

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

[HBKC11] Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. The SSL landscape: a thorough analysis of the x.509 PKI using active and passive measurements. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 427–444. ACM, 2011.

[Hen12] Nadia Heninger. New research: There's no need to panic over factorable keys–just mind your Ps and Qs, 2012. https://freedom-to-tinker.com/blog/nadiah/new-research-theres-no-need-panic-over-factorable-keys-just-mind-your-ps-and-qs.

[HPSP10] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *Security & Privacy, IEEE*, 8(6):40–47, 2010.

[IEC08] Quantities and units – Part 13: Information science and technology, 2008.

[IMD] The internet movie database. http://www.imdb.com/stats.

[jGSMB03] Eu jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. SiRiUS: Securing remote untrusted storage. In *Proc. Network and Distributed Systems Security (NDSS) Symposium 2003*, pages 131–145, 2003.

[JKJ07] Ari Juels and Burton S Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597. Acm, 2007.

[Joh99] Don B. Johnson. ECC, future resiliency and high security systems. Certicom Whitepaper, 1999. www.comms.engg.susx.ac.uk/fft/crypto/ECCFut.pdf.

[KAF+10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit RSA modulus. In Tal Rabin, editor, *Crypto 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 333–350. Springer, Heidelberg, 2010.

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

[Len87] Hendrik W. Lenstra Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126(3):649–673, 1987.

[Len98] Arjen K. Lenstra. Generating RSA moduli with a predetermined portion. In Kazuo Ohta and Dingyi Pei, editors, *Asiacrypt '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 1998.

[LHA+12a] Arjen K Lenstra, James P Hughes, Maxime Augier, Joppe W Bos, Thorsten Kleinjung, and Christophe Wachter. Public keys. In *Advances in Cryptology–CRYPTO 2012*, pages 626–642. Springer Berlin Heidelberg, 2012.

[LHA+12b] Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter. Ron was wrong, Whit is right. Cryptology ePrint Archive, Report 2012/064, 2012. http://eprint.iacr.org/.

[LL93] Arjen K. Lenstra and Hendrik W. Lenstra Jr., editors. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1993.

[LM10] M. Lochter and J. Merkle. Elliptic curve cryptography (ECC) brainpool standard curves and curve generation. RFC 5639, 2010.

[LN11] Daniel Loebenberger and Michael Nüsken. Analyzing standards for RSA integers. In Abderrahmane Nitaj and David Pointcheval, editors, *Africacrypt '11*, volume 6737 of *Lecture Notes in Computer Science*, pages 260–277. Springer, 2011.

[MAL15] Hugues Mercier, Maxime Augier, and Arjen K Lenstra. Step-archival: Storage integrity and anti-tampering using data entanglement. In *Information Theory (ISIT), 2015 IEEE International Symposium on*, pages 1590–1594. IEEE, 2015.

[Mit04] Michael Mitzenmacher. Dynamic models for file sizes and double pareto distributions. *Internet Mathematics*, 1(3):305–333, 2004.

[MM02] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[Moo08] H. D. Moore. Debian OpenSSL Predictable PRNG Toys. See http://digitaloffense.net/tools/debian-openssl/, 2008.

[NS02] Phong Q. Nguyen and Igor Shparlinski. The insecurity of the digital signature algorithm with partially known nonces. *Journal of Cryptology*, 15(3):151–176, 2002.

[NS03] Phong Q. Nguyen and Igor Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Design, Codes Cryptography*, 30(2):201–217, 2003.

[PGES05] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *Peer-to-Peer Systems IV*, pages 205–216. Springer, 2005.

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

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

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[SGMV09] Mark W. Storer, Kevin M. Greenan, Ethan L. Miller, and Kaladhar Voruganti. POTSHARDS: a secure, recoverable, long-term archival storage system. *Trans. Storage*, 5(2):5:1–5:35, June 2009.

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

[SNS] Gottfrid Svartholm, Fredrik Neij, and Peter Sunde. The Pirate Bay. http://thepiratebay.se.

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

[T+] Jan Tomášek et al. Blacklisted moduli. See http://mirror.switch.ch/ftp/mirror/debian/pool/main/o/openssl-blacklist/ and http://pocitace.tomasek.cz/debian-randomness/index.html.

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

[U.S09] U.S. Department of Commerce/National Institute of Standards and Technology. Digital Signature Standard (DSS). FIPS-186-3, 2009. http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf.

[VFBH11] Nevena Vratonjic, Julien Freudiger, Vincent Bindschaedler, and Jean-Pierre Hubaux. The inconvenient truth about web certificates. In *The Workshop on Economics of Information Security (WEIS)*, 2011.

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

[Wie92] Michael J. Wiener. Personal communication, 1992.

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

[WOW08] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: the least-authority filesystem. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 21–26. ACM, 2008.

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

[YRS+09] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: results from the 2008 debian OpenSSL vulnerability. In Anja Feldmann and Laurent Mathy, editors, *Internet Measurement Conference*, pages 15–27. ACM, 2009.

[Zim12] Paul Zimmermann et al. GMP-ECM (elliptic curve method for integer factorization). https://gforge.inria.fr/projects/ecm/, 2012.

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

MAXIME AUGIER
Av. du Tir-Fédéral 48A, CH-1024 Ecublens
Phone: +41 76 578 44 36
E-Mail: maxime.augier@gmail.com

Swiss
December 4th 1983
Single

OBJECTIVES AND INTERESTS

Complete my PhD in the field of secure cloud storage systems. Interests in network engineering, radio communications, and cryptography.

EDUCATION

2004-2008 Swiss Federal Institute of Technology – EPFL, Lausanne, Switzerland
Master in Communication Systems , graduated in 2008

2001-2004 Swiss Federal Institute of Technology - EPFL, Lausanne, Switzerland
Bachelor in Communication Systems, graduated in 2004.

1994-2001 Florimont Institute, Geneva, Switzerland
Federal Maturity, scientific section. Graduated in 2001.

PROFESSIONAL EXPERIENCE

11.2008 - current **Cryptologic Algorithms Laboratory / EPF Lausanne – Doctoral Assistant**

- Working on my thesis on secure, anonymized, censorship-resistant cloud storage
- Administration of the laboratory's Linux-based cluster
- Designing and teaching practical labs for Pr. Janson's IT Security Engineering course
- Teaching of various beginner and advanced programming courses (C, C++, Perl, Java)

8.2008-11.2009 **Adeya S.A. – Products Engineer**

- Validated the cryptographic design of a secure mobile telephony system (Software-based and running on commodity hardware)
- Designed encrypted VoIP infrastructure software for scalability and redundancy
- Systems administration for a dozen of Linux physical and virtual servers

3.2005-9.2005 **Indiana University – Researcher**

- Implemented a proof-of-concept implementation of the Delayed Password Disclosure protocol from Jakobsson & Myers

9.2003-9.2006 **Ilion Security SA, Geneva, Switzerland – Auditor**

- Participated in several security audits of high-profile information systems,
- Developed various intrusion software and tools implementing proprietary techniques, especially covert channel communication tools (TCP-over-HTTP, over-DNS, ...)

4.2003-4.2008 **PolyLAN – Network Administrator**

- Developed an innovative strong network authentication and security system
- Hands-on experience with SME grade networking hardware
- During the events, managed the internal network of up to 400 guest hosts, potentially hostile.

7.2001-9.2001 **United European Bank, Geneva, Switzerland – Intern**

Developed software for:

- Archiving of recommandations from the financial analysis team, and generation of reports for the customers and portfolio managers
- Major achievement: found and fixed an unnoticed bug leading to some financial statistics being incorrectly computed and published
- Monitoring of customer portfolios, matching with pre-established risk profiles, and automatic adjustment to the desired profile
- Tracking and rating performance of the portfolio managers

PUBLICATIONS

- **SteP-archival: Storage integrity and anti-tampering using data entanglement,** H. Mercier, M. Augier, A.K. Lenstra
  IEEE International Symposium on Information Theory (ISIT) 2015

- **Public Keys,** A. K. Lenstra, J. Hugues, M. Augier, J.W. Bos, T. Kleinjung, C. Wachter
  Advances in Cryptology – CRYPTO 2012

- **Parity-Regular Steinhaus Graphs,** M. Augier, S. Eliahou,
  Mathematics of Computation Vol. 77 (2008)

- **Phishing and Countermeasures** (contributor), Edited by M. Jakobsson & S. Myers. Wiley, 2007

ACADEMIC WORK

Winter 2008 **PlayStation 3 Cluster** – Cryptographic Algorithms Laboratory, EPFL
- Set up a frontend system for a 200 nodes cluster (Linux-based, for storage and job queue management), network planning and wiring, installation of a monitoring system, wrote some environment-specific tools – http://ps3.epfl.ch

Supervisor: Dag Arne Osvik – dagarne.osvik@epfl.ch

Winter 2007 **Efficient Wired Networks for Untrusted Hosts** – Lab. For Communications and Applications, EPFL
- Design and implement a strong authentication solution, with low configuration and performance overhead, for unmanaged hosts in wired networks. The solution was successfully deployed during the three next PolyLAN events.

Supervisor: Dr. Panos Papadimitros – panos.papadimitros@epfl.ch

Summer 2005 **Delayed Password Disclosure** – School of Informatics, Indiana University
- Implemented a portable proof-of-concept implementation of the Jakobsson-Myers Delayed Password Disclosure protocol

Reference: Pr. Markus Jakobsson – markus@indiana.edu

Winter 2004 **Network Coding Simulator** – Lab. For Communications and Applications, EPFL
- Implement a simulator to benchmark network coding performance on a 2D broadcast wireless network. Written in C++ using NTL for the network code implementation, with some Perl scripts for post-processing.

Supervisor: Dr. Joerg Widmer – widmer@acm.org

Winter 2003 **WaJE, Web Application for Junior Entreprises** – Programming Methods Laboratory, EPFL
- Design a web application for administrative management of Junior Entreprises, using modern standards and design patterns (J2EE, PostgreSQL backend, Aspect-like extensibility)

Supervisor: Dr. Buraq Emir – buraq.emir@epfl.ch

LANGUAGES

French: native
English: fluent (CAE, June 2004)
German: scholar notions

ACTIVITIES AND INTERESTS

Regular contributor of the FIXME hackerspace in Lausanne, CH
Amateur radio: CEPT Licensed radio operator since March 2008 (Callsign HB9EKD)
Sports: Snowboarding