Long term data storage using peer-to-peer technology

Paulus Nicolas Meessen
Computing Science
Radboud University

Supervisor: dr. Jaap-Henk Hoepman
Second Reader: prof. dr. Arjen de Vries

August 18, 2017

Abstract

This project investigates the possibility of creating a privacy-friendly file-storage consumer product suitable for long-term storage. A theoretical background is given for the current state of file sharing technologies, theories on privacy, technology to protect privacy, and peer-to-peer mechanisms. Based on this knowledge, we composed a list of requirements to design the protocol.

The protocol by Osipkov et al. was chosen as the starting point since it is a distributed system and robust against free-riders. To safeguard the privacy of the users, we improved the protocol so that it can operate on an anonymity network. We created two simple schemes based on Merkle Trees that allow third party monitoring of the availability of remotely stored files.

Privacy, security and data protection are essential to consumer technology. Based on what we developed, we show that it is possible to create a privacy-friendly file-storage consumer product for long-term storage, robust against free-riders.

Contents

1 Introduction

Currently there is no privacy preserving and user friendly consumer product that is suitable for long-term online file storage. There are many cloud storage consumer products that are cheap and user friendly, however these are privacy invasive since they can access the private data of their customers. The objective of this project is to investigate if it is feasible to create a storage product that does not depend on a central authority for storage, preserves the privacy of the users, and remains easy to use.

Privacy is an essential human right (Westin, 1967). Yet we struggle to find a good description or definition of what privacy actually is (DeCew, 2015). At the same time we do notice that the right to privacy is affected by new technology and media. Through the lens of these new technologies - photography, newspapers, and these days data collection through the internet - we have developed ideas about what privacy could be. These ideas on privacy usually describe the level of control people have over their data. The contemporaty interpreation of privacy comes from Nissenbaum in a response to the information technology that has become part of our (online) social life . Nissenbaum describes privacy as right to integrity of context (Nissenbaum, 2009). Protecting privacy should involve helping people to be in control of the context in which their data flows. Protecting digital privacy therefore is about more than just access control to the data.

To illustrate that access control is not sufficient we can consider messaging applications. Even though the popular chat application WhatsApp[1] provides end-to-end encryption of the messages[2] to hide all the data, it remains a very valuable source of personal information for its parent company Facebook. Facebook can still use all the metadata generated by the network to build a social graph of personal information. Users have no control about the context in which Facebook uses their personal data.

We are often not aware about the flow between different contexts of our personal data. When telling a friend a secret, the context dictates that the friend will likely not share this information with others. If the friend does share the secret, you would feel that as an infringement on your privacy. When we use online services, we often share highly personal information, without being aware of this. This can be as innocuous as looking up a medical ailment online. Even though we see the advertisements on such pages, we rare consciously experience this as an infringement on privacy that this personal information about us was shared with another party. It is not always easy to recognize our data flowing from one context to another.

When building products that can respect the users' privacy it is important to hide the metadata. In a private context, the metadata *that* you have talked to a doctor can be as much of an infringement to your privacy as when it would be public *what* you discusses with your doctor. The metadata is a very large component that dictates the context in which it can be perceived.

Hiding the metadata imposes constraints on the design. The price for solving the problem under these constraints is not only payed in hardware or infrastructure, but also in terms of a poorer user experience (Abu-Salma et al., 2017). PGP is the most well known tool for email encryption. The complicated user experience of PGP is often considered the main reason why the tool has not been adopted by the mainstream of users to encrypt their e-mail messages (Abu-Salma et al., 2017). As a result hardly anyone uses end-to-end encryption for e-mail messages. User friendliness therefore cannot be disregarded when building new privacy tools, and should be seen as a fundamental requirement.

This idea is part of the notion of privacy friendliness in which users are aware and in control of the way technology affects their privacy.

[1]`https://www.whatsapp.com/`

[2]`https://whispersystems.org/blog/whatsapp-complete/`, accessed 2017-07-22

Proper security is essential for privacy and the control of our data. Security even become a selling point in consumer technology as of late. Almost all the large chat applications have implemented end-to-end encryption. The billion users of WhatsApp all use end-to-end-encryption. There even are a number of chat protocols that attempt to achieve privacy friendliness for chatting - such as Ricochet (`https://ricochet.im/`) or Vuvuzela (Van Den Hooff et al., 2015).

Unfortunately the world of online file storage still is behind on chat applications, not just about privacy but even regarding security. Storage products generally do not even protect the confidentiality of the customers' data. Dropbox (`https://www.dropbox.com/`, one of the larger companies for consumer cloud storage, has yet to implement end-to-end encryption for the data stored online. This allows companies like Boxcryptor (`https://www.boxcryptor.com/`) to enter the market with a product that provides end-to-end encryption within the infrastructure of other products such as Dropbox. If storage companies earn money through advertisement and profiling based on our personal files and data, then they do not respect the context of our data. This is a huge infringement on privacy.

We intend to design a privacy-friendly decentralized (peer-to-peer) storage protocol to be used by regular consumers. This storage is for off-site and redundant copies of personal files and data. It therefore ought to compete with external hard drives, Network Attached Storage devices (NAS) and the various cloud storage solutions mentioned in subsection 2.1. After the consumer has bought and configured the product, no interactions should be required to perform its function during the lifetime of the hardware.

There are three main challenges in this project. The first two are the requirements for providing a basic decentralized data storage protocol. This protocol should be able to deliver long term data storage. This protocol should provide a measure of the availability of the stored data. Combined, these two requirements should result in the trust of the consumer that the protocol will deliver on its functionality, and return all stored data when requested. The final challenge is to provide privacy in the protocol. This protocol should perform its function without revealing any information about the user, not through the data stored in the network and not through the metadata produced.

In section 2 we first examine a number of existing file-storage solutions. Then we will explore in subsection 2.2 peer-to-peer technology and how we can use it to do file storage without having a central authority. In addition we will consider the general topic of privacy and the methods that are available to provide privacy in peer-to-peer networks (Subsection 2.3). The methods and incentives that allow us to regulate the behavior of users in a peer-to-peer network are also important for the long term viability of a protocol, and are described in subsection 2.4.

Based on this knowledge we specify the requirements for a protocol for privacy friendly long-term storage in section 3. From these requirements we describe in section 4 the protocol presented by Osipkov et al. as a possible framework to be built upon (Osipkov et al., 2006). The Osipkov et al. protocol provides fairness among peers. Fairness protects against the threat of free-riders that consume disproportional resources from the network. Fairness therefore allows for meaningful agreements about long-term storage. However, the Osipkov et al. protocol does not yet meet all our requirements. In particular it should run on an anonymity network, so that it can be privacy preserving. The core qualities of the protocol mostly remain unaffected by this change. It does become more prudent to provide the peers with a tool that allows verification of the availability of remote files at the anonymous peers. This paper contributes two simple probabilistic schemes, outlined in subsection 5.2, to test the availability of encrypted remote files without requiring the querying party to possess these files.

2 Technical background

This section provides some of the theoretical background used in this project. In order to set proper requirements for the protocol, we first consider a number of existing protocols. Then we will look at peer-to-peer networking, and reflect on some of the strengths and challenges. Thereafter we will give a general introduction to the topic of privacy, focusing on how law and technology can help to create privacy preserving technology. Since networking is inherently social, we also review the economies and mechanism that exist to incentivise the peers in a peer-to-peer network. The last part of this section discuss the Merkle Tree, which is core to our contributions in testing the availability of remote data.

2.1 Existing online storage technologies

Currently the market for online data has been saturated by cloud storage providers. As a general overview, we will list a number of existing, non-peer-to-peer, online file storage solutions. This provides a landscape for the data-storage consumer products and experience. The list below is loosely based on their ability to be on the first-page of Google results for 'secure online storage'.

- **Cloud storage products**, such as Dropbox (`http://dropbox.com`) or SURFdrive (`https://www.surfdrive.nl/`), allow users to synchronize their files across devices and get to them via a web interface. On many devices, users interface with the storage by means of a virtual directory in their file system. Along the ability to have online and offline redundancies, cloud storage often allows users to share directories with each other. Some products are designed for a specific purpose such as backup, for example Tarsnap (`https://www.tarsnap.com/`) or Backblaze (https://www.backblaze.com/). Consumers generally only pay for the storage they use, not for the bandwidth.

- **Add-on security software**, from companies like BoxCryptor (`https://www.boxcryptor.com/en/`), sell client-side software that provides improved security features to existing storage products, such as end-to-end encryption of user data. There is a demand for these features, yet not all big storage product offer these. Therefore, there exists a market for this add-on software.

- **Cloud storage providers**, such as Amazon AWS's S3 (`https://aws.amazon.com/s3/`), provide generic online data storage. They provide the infrastructure for many storage products. Users pay for the infrastructure based on the storage they use and for the bandwidth they consume.

- **File hosting websites** should also be considered since they compete in the market for (peer-to-peer) file-sharing. File hosting websites differ from the cloud storage products in the sense that they are usually about single files, offer a public interface for accessing the file, and optimize for convenient sharing. In general these are are advertisement-sponsored pigeonholes for file storage. Most (in)famous is MEGA (`https://mega.nz/`), that now promotes end-to-end encryption of the data and even quotes the Universal Declaration of Human Rights, Article 12: the right to privacy. The Dutch company WeTransfer (`https://wetransfer.com/`) is a good example of a more mainstream product.

Since the nineties there have been numerous research projects on peer-to-peer networking. A number of those have focused on file sharing and some also addressed file storage. Almost none of these technologies are currently on the market (yet). An increase of the research activities between 2000 and 2008 has yielded some novel ideas for storage. One of these was the peer-assignment scheme of Osipkov et al. that is addressed in section 4. Research on peer-to-peer file sharing has been ongoing, in particular for its use in content and media delivery systems (Pouwelse et al., 2008). Research into peer-to-peer storage has recently seen renewed interest with the surge of blockchain technologies. We have listed a number of large or interesting peer-to-peer projects that have made it to market below.

- **Tahoe-LAFS** (`https://tahoe-lafs.org/trac/tahoe-lafs`) is a decentralized cloud storage system (Wilcox-O'Hearn and Warner, 2008). It also has a fork that runs anonymously on I2P. Users contribute the hardware and bandwidth to the network. Some choose to outsource this via cloud storage providers. Least Authority (https://leastauthority.com) is the spin-off company from the creators of Tahoe-LAFS that - for a monthly fee - adds cloud storage to the network on their customers behalf.

- **Blockchain based file storage** has two main varieties. The first tries to replace the proof-of-work for mining coins on a blockchain by a proof-of-storage. This idea has been suggested as Permacoin (Miller et al., 2014) and Spacemint (Park et al., 2015). Blocks are mined by proving the possession and integrity of some subset of the dataset that as a whole is too large for a single (subset) peer to store. This proof-of-possession then consists of a public verifiable digest of this dataset. Both Permacoin and Spacemint want the storage of the entire dataset to be valuable in itself. For example, the dataset could consist of the entire digitized collection of a library. Such datasets generally do require a central authority to curate and authenticate them.

  The mechanisms can also work without a central authority. The entire dataset could be generated through some mathematical function that will not allow on-the-fly lookup without storing all the output. Since this system does not allow storage, we refer to the mechanism as proof-of-capacity. Burstcoin[3] (Wikipedia, 2017a) mined their blocks based on proof-of-capacity and was operational since 2014. Although we intend to actually storing something meaningful, some of the technology could be useful.

  The second variety of blockchain based file storage merely leverages the cryptocurrencies that arise from mining a blockchain to broker between people offering storage and people wanting to buy storage. Examples of this are Storj ((Wilkinson et al., 2016)) or Filecoin (`https://filecoin.io`). The company Minebox (`https://minebox.io/`) already sells a hardware NAS that can automatically rent out storage to these markets to earn cryptocoins.

Note that many of the companies mentioned are established in the United States. Although they sometimes have data-centers in Europe, this does not mean that they adhere to the stricter European privacy laws.

2.2 Peer-to-peer Technology

In this subsection we discuss the possibilities, advantages and challenges of the peer-to-peer technology that will be used to create our file storage protocol.

Peer-to-peer computing or peer-to-peer networking is a subtype of distributed computing where all the nodes that are part of the network are considered equal.

Networked computing is typically described as a master-slave or server-client relation. The different nodes can have different tasks, different privileges and different software. The relation between nodes therefore is asymmetrical, and the network will be organized around a central node, like the musicians of an orchestra around the conductor.

[3]Burstcoin seems to have seized operation.

When many nodes need access to a single resource or privilege, a number of problems may occur. For example, if nodes send many requests for an identical result, they all put a demand on the computational resource or bandwidth and as an consequence the performance and quality of the service may suffer. The network may congest and if the node is part of the critical section of a computation this can create a single-point-of-failure. Having a single-point-of-failure is a security concern.

To mitigate these problems the computation can be designed to run decentralized. This means that there should not be a single point of failure and the demand for resources can now be balanced across multiple nodes. This is like a central government allowing provinces and municipalities to build their own roads. However, all the previous problems remain if the instances still depend on a single shared resource or authority.

When the task of the nodes becomes so ubiquitous that they no longer depend on special resources or a centralized authority, it becomes possible to move towards a more symmetrical relation between the nodes. Some networks are so sparse and segmented that nodes need to be able to sustain without having access to specific parts of the network and therefore require more symmetrical and self-sufficient nodes. Imagine building a networked application that also should work for future astronauts on Mars: your nodes are going to be very far apart. The main latency is always going to be caused by the approximately half-light-hour distance round-trip of the messages. It hardly matters how much computing infrastructure you build on earth, the sparseness of your nodes through space impose a physical limit. In such a cases it becomes almost essential to run a computation fully distributed. *Every* node can consider *any* other node of the system as a peer and request the service from this user.

(image by 1983~enwiki, 2017, CC-BY-SA-2.5)

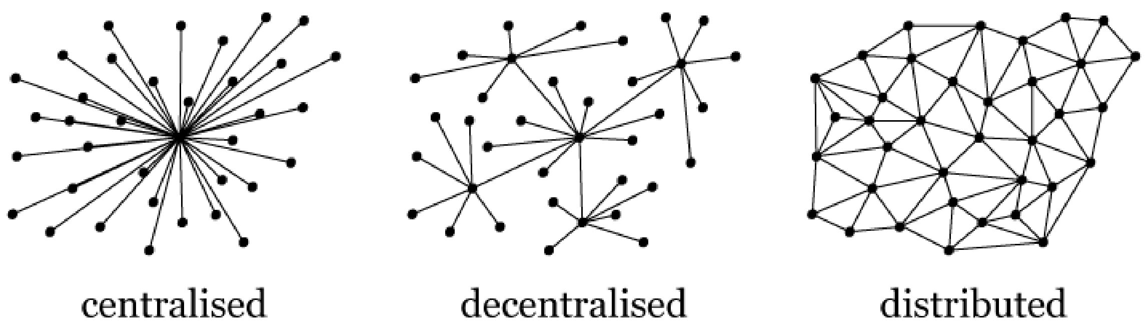

Figure 1: Different network models

Peer-to-peer networks perform fully distributed computations where each node has the same functional capabilities as any other node.

Running a computation on a peer-to-peer network provides even more robustness against single points of failure and has even more nodes to balance the demand for resources. At the same time, the lack of organization and the lack of (central) control poses new challenges for the entire network to organize itself. Creating consensus in a distributed network is already considered a hard problem (Lamport et al., 1982). In some cases, for example due to lack of synchronization or in the presence of malicious and unpredictable peers, consensus has been proven an impossibility. (Fischer et al., 1985) Organizing a peer-to-peer network in the presence of malicious peers is therefore not a simple task.

Useful behaviors can emerge from node-to-node interactions in a network with malicious peers. Blockchain technology, as has been popularized by the pseudonymous Nakamoto with Bitcoin, leverages the mutual distrust of the peers in a network into a shared effort from which a collectively agreed upon consensus emerges.

For this project we are mainly interested in ways to organize and program the nodes such that they - as a whole - participate in the storage. A number of simple tasks to structure networks, such as leader election, mutual exclusion, and self stabilization are well understood and form the building blocks for more advanced applications. (Lynch, 1996) Locating resources or peers in a distributed network by means of a Distributed Hash Table (DHT) is the most common protocol used for peer-to-peer applications.

2.2.1 Distributed Hash Tables

A hash table is a data structure that allows the user to `INSERT`, `DELETE` and `SEARCH` records in a table and is optimized to access a record from the table efficiently based on the key. (Rivest and Leiserson, 1990) If we want to search for a record in a regular table, first we have to sort through the data and then filter until we locate the record.

If there is plenty of memory available, search in a table can be optimized. Instead of adding a new record for every new item like in a regular table, in a hash table the space for all the possible keys is allocated in advance. New records are then placed directly in the space reserved for their key. For example, when making a 'reverse phonebook' for caller-id: all the possible phone numbers are listed, and for every new caller the name is written next to the phone number. Looking up the name belonging to a phone number (key) is as easy as going directly to that number in the list. This only takes $O(1)$ steps. However, not all data have such nice natural keys.

In order to create keys for all records in our table we make use of a hash function. A **hash function** is a one-way function that maps an arbitrary large input set onto a (smaller) finite set. Most applications of hash functions require this onto-mapping to have a uniform distribution. A simple view of what a hash function does, is that for any key it assigns the record to a specific bin. The uniform distribution then requires that on average equally many inputs get placed in each bin. In order to locate a resource we now only have to look in the bin that belongs to our key.

If you do not have enough memory to allocate a bin for every possible key, you can use a hash table to map a large space of keys onto the smaller set of bins. A trade-off has to be made to deal with hash collisions. A collision occurs when the hash function maps two different keys onto the same bin.

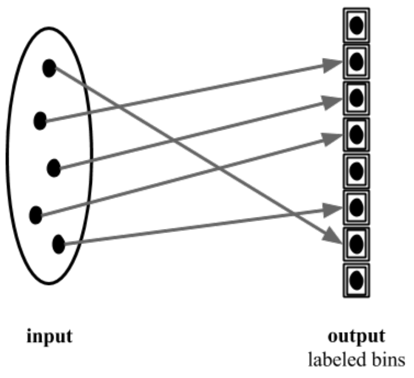

Figure 2: hash function

There are three main solutions for dealing with collisions. The easiest is to just throw away the data in event of a collision. This of course is only acceptable if we don't mind losing some data. The second solution is to re-hash the colliding key found in the bin, and moving towards a new empty bin. This approach makes optimal use of the available memory but as it fills up the table, the lookup performance moves from $O(1)$ to $O(n)$ steps and eventually runs out of available space. The final solution is to attach a link to a new data structure once a collision occurs. This means that for the most part the hash table still has instant lookup performance and in the case of a collision, it will have the performance of the associated data structure. These data structures are likely less efficient than the hash-tables. However, the number of records in this associated data structure is going to be at least an order smaller than the total number of records. Lookup in the associated data structure is going to be relatively fast. Therefore, the lookup performance through all the data will be acceptable. For example, it does not matter in a pocket phonebook that the names are not sorted beyond the first letter of the last name, since at each of these letters you will find about 10 names from which you can find the right one at a glance.

The nodes in a distributed network often are connected according to some social structure in the real world. Computers often are where people are. We want to impose a data structure on the network in such a way that we leverage the Small World Phenomenon. The Small World Phenomenon, as made famous by Travers and Milgram, states that in the (socially) connected graph of people in the world every person is connected to any other person through a link of at most six people. If we can create a hash table on top of a distributed (social) network we might be able to combine the Small World Phenomenon with the $O(1)$ lookup performance of a hash table and have efficient resource location in peer-to-peer networks.

This is what Distributed Hash Tables like Pastry (Rowstron and Druschel, 2001) and Chord (Stoica et al., 2001) can achieve. In the final part of this section we are going to look at the Chord protocol and how it facilitates $O(\log N)$ key-value retrieval on a peer-to-peer network (Stoica et al., 2001).

In Chord the `SHA-1` hash function is used to give every node an identity number (`ID`) between 0 and $2^{160}-1$. Since `SHA-1` is a uniform hash-function with 160-bits output, and the number of nodes (N) is a lot smaller than $2^{160}-1$, we expect every node to receive a unique integer as `ID`. Next, all nodes are organized in a ring, where every node stores the `ID`'s of the two nodes that are closest to it $\pmod{2^{160}}$. Every node now is capable of determining if they are have the closest `ID` to some other integer $\pmod{2^{160}}$. Now it is a hash table, where the `SHA-1` hash of a *key* can be used to find an integer, for which the node with the closest `ID` has to store the associated value.

However, this is a very inefficient method, as the messages would have to pass $O(N)$ steps through the entire ring until it gets to the correct node. Ideally every node would maintain a list of all the other `ID`'s such that the lookup can be done locally. As the number of nodes (N) usually is very large, it is very costly to store such a table at a single node. A compromise is to have each of the nodes store a table with only a small number of strategically stored `ID`'s that allows for efficient routing towards the the correct node.

The table uses for this purpose in Chord is called a *finger table*. A node n stores m fingers in the finger table, each of which point at a live node. The ith finger, starting at $i = 1$, in the table points to the live node whose `ID` succeeds ($n + 2^{i-1} \mod 2^{160}$). The fingers are an exponential series of `ID`'s that on average can bring the search one order closer to the correct node. For a network of N nodes, on average it takes $\log N$ requests to get to the correct node.

Creating such a table is not without cost. Whenever a node joins or leaves the network the finger tables need to be updated. As a upper bound for the complexity in a network of N nodes, on average every node has to check if one entry in their finger table is still correct at the cost of $\log N$ queries per node, resulting in $O(N \log N)$ steps. This can be optimized to $O(\log N)$. (Stoica et al., 2001)

The Chord example shows that we have a method for organizing a peer-to-peer network using the Distributed Hash Table, with the performance of $O(\log N)$ queries per lookup. Peer-to-peer networking therefore becomes suitable as the networking backbone in our application of file storage.

2.2.2 Current peer-to-peer file sharing applications

File-sharing has always been the most wide known application of peer-to-peer technology. Most notorious was Napster by Shawn Fanning and Sean Parker, which allowed searching and sharing music files (Wikipedia, 2017d). Following the popularity, more peer-to-peer networks were created such as GNUnet (`https://gnunet.org/`). All these early technologies are built on distributed hash tables.

File-sharing - in particular the free sharing of otherwise copyrighted and payed content - really took of with the introduction of BitTorrent (Protocol specification `http://www.bittorrent.org/beps/bep_0003.html`, accessed 24-07-2017). When downloading a file, BitTorrent allows a user to become a server of the parts of the file that it already has obtained, and advertises this in the network. In a centralized system the files generally become unavailable if the number of requests grow beyond what the server can handle. In BitTorrent the opposite occurs: the more popular the file, the more peers contribute to its distribution and the more available it will become. Even though BitTorrent traffic is in decline[4] as illegal file-sharing is replaced by legal alternatives, the technology is still used in new projects, for example by CacheP2P (`https://www.cachep2p.com/`) in an effort to create performant decentralized Content-Delivery-Networks.

2.3 Privacy

'The term privacy is used frequently in ordinary language as well as in philosophical, political, legal and technical discussions. Yet there is no single definition or analysis or meaning of the term.' (DeCew, 2015) One of the early and influential ideas on privacy comes from the writings of Aristotle. Aristotle describes privacy as the border between the public sphere of political activity and the privacy sphere of home life. While the definition of both political activity and home life have changed significantly since antiquity, this idea still has a lot of influence on the modern idea of privacy. For example the European Convention for the Protection of Human Rights and Fundamental Freedoms from 1950 presents the right to privacy in Article 8 as the *Right to respect for private and family life.* However, with the advent of modern media, and in particular in the past few decades with the increased integration of internet into society, privacy has had numerous new interpretations and definitions.

The first major media innovation that helped to shape the modern views on privacy was the mass adoption of newspapers and photography in the late nineteenth century. These innovations challenged longstanding ideas on which agents could affect the distribution of one's personal identity. The work of Warren and Brandeis let to the interpretation of privacy as *the right to be let alone.* (Warren and Brandeis, 1890) This was the foundation of what later was going to be the concept of control over information about oneself. (DeCew, 2015) It also demonstrates that often multiple parties have a responsibility to enforce the right to privacy, and is not something bound to the individual.

In 1967, Westin published a book on Privacy and Freedom, in which he investigate the universality of privacy in a response to the cold war spy technology. Furthermore, studies of animals had demonstrated that a desire for privacy is not restricted to humans. Westin presented privacy as the *claim of individuals, groups and institutions to determine for themselves, when, how and to what extent information about them is communicated to others.* This furthered the ideas on controlling one's personal information as a means to preserve the right of privacy.

With the mainstream adoption of the internet at the start of the twenty-first century, a new technological threat to privacy appeared. Collections of personal data were stored in databases around the world and were being traded and combined to mine new insights into individuals and groups. Agre and Rotenberg pose privacy as *the Freedom from unreasonable constraints on the construction of one's own identity*, as was feared that people would no longer be in control of what (computer) systems decides about their identity. (Agre and Rotenberg, 1998)

In 2009,Nissenbaum was able to contribute to the discussion of privacy, by interpreting privacy as the *right to integrity of context.* (Nissenbaum, 2009) Contextual integrity is a

[4]`https://www.digitalmusicnews.com/2013/11/12/illegalfilesharing/` accessed 24-07-2017

useful framework to pinpoint why some transactions of personal data can be experienced as an infringement to one's privacy. For example, when someone shares information with a physician in a medical context, it is generally acceptable when the physician shares this information with another medical specialist. The context in which this information has been shared remains the same. However, if the physician would decide to sell this information to an advertisement agency, this is experienced as extremely intrusive to one's privacy. The same framework of context can be applied to businesses on the internet. Going to a medical website and being shown an advertisement is so common that it is very difficult to notice the different context from going to a shoe website to buy shoes. Nissenbaum forces us to see the implications for privacy by requiring the identification of the context.

All these ideas have been integrated into the societal thinking about privacy. Both law and technology are instruments to help preserve our right to privacy. We have already seen the inclusion of the right to privacy in the European Convention for the protection of Human Rights and Fundamental Freedoms, but there are many more legal instruments to help protect privacy and make sure that people can be in control of their own data, identity or context (Council of Europe, 1950). Notable laws to protect privacy in European context are the Data Protection Directive (Directive, 1995) and its successor the General Data Protection Regulation (EC Reg 2016/679, 2016). It must be remarked that there is a distinction between privacy and data protection. While the two go hand-in-hand, having your data protected in compliance to the law does not imply that no infringement has occurred on your privacy. The other way around; even if some party manages to respect privacy, it does not immediately imply they are compliant to data protection law. The enforcement of the laws regarding privacy are an essential element to actually achieving privacy.

The other instrument we have to protect our privacy is technology, and the practices we have to build these technologies. There is a growing industry for Privacy Enhancing Technologies and Privacy by Design as a technical requirement will be part of the General Data Protection Regulation (EC Reg 2016/679, 2016). This does create some difficulties for both lawyers and engineers. The law does not give a clear definition for Privacy by Design, and since it is a relatively new requirement a clear and shared definitions is still lacking. Some footholds exists, such as the Privacy Design Patterns (Hoepman, 2014) or the OWASP Top 10 Privacy Risks Project[5]. However, Privacy By Design still needs more actual tools and practice, such that the relevant legal agencies can learn which meaningful practices should be enforced.

An additional technology for protecting privacy through data protection is the use of anonymous communications. When the context of data can be hidden, it gives back control to the original owner of the data. Providing anonymity is not easy. Even though we have usable mathematical descriptions of various approaches to anonymity, the practical experience is often that there are additional channels through which an identity can leak. This is why the use of pseudonyms almost never implies actual anonymity.

The formal description of anonymity that is considered for this document is (Pfitzmann and Hansen, 2005). Pfitzmann and Hansen define anonymity through set theory: "Anonymity is the state of being not identifiable within a set of subjects, the anonymity set". This still allows for a number of degrees of anonymity. In a weaker reading of this definition, anonymity implies that the likelihood of any other subject in the anonymity set being identified as the sender or receiver of some message is significant. In the stronger reading, every subject in the anonymity set should always have an equal likelihood of being the sender or receiver of some message.

As a technical requirement anonymity often implies unlinkability. "Unlinkability of two

[5]`https://www.owasp.org/index.php/OWASP_Top_10_Privacy_Risks_Project`, accessed 13-08-2017

or more items of interest [...] means that within the system [...], the attacker cannot sufficiently distinguish whether these [items of interest] are related or not.". A common measure for such anonymity is by implementing k-anonymity. (Sweeney, 2002) k-anonymity on a dataset is the requirement that in any ordered subset of the dataset, k is the lower bound for the number of records that exist in such a set.

2.3.1 Protecting privacy in peer-to-peer networks

Peer-to-peer networking has a valuable application in anonymous communication technologies. For communication to be anonymous it should be impossible to relate a message to both its sender and recipient based on the metadata of that message. Large adversaries, like governments or foreign intelligence agencies, are said to have the power to observe all the network traffic. This makes it verify difficult to guarantee anonymous communication.

Encryption allows us to create 'envelopes' that can only be opened by the intended recipient. But if the mailman is evil: how will we send our messages without being able to write the address on the outside of the envelope?

This can be achieved by sending messages through a trusted intermediary, called a mixer, that 'washes' the details about the sender of the message.

In order to send an anonymous message, you put this message in an envelope with the final address on it, and then put this envelope inside of another envelope that you address to the mixer. An adversarial mailman cannot open these envelopes, so just delivers them to the mixer. The mixer opens her mail and finds new envelopes. Once the mixer has received a certain number of envelopes, she puts them in a large container and *mixes* the container until it is impossible to determine the original order of the envelopes. The mixer then gives all the envelopes to the mailman. The mailman cannot know who the original sender of an envelope was, but now has an address to deliver the message. If we can trust the mixer, we can use the mixer to send anonymous messages in the presence of an adversarial mailman.

The anonymity set will consist of all the people who send their message to the mixer and were in the same container. This can be a relatively small set, in which case it might be possible to locate the sender. To increase the size of the anonymity set it is possible to chain a number of mixers. This does mean that there are more parties that need to be trusted. Chaum argues that chaining mixers still can achieve anonymity if at least one mixer in the chain can be trusted (Chaum, 1981) .

Currently the most used method for anonymous communications is Onion routing. Onion routing assumes that every peer in a network has a known public key. By encrypting the message multiple times using a sequence of public keys, the user effectively wraps the message in multiple layers of encryption. Peers in the network can 'peel off' the outer layer of encryption, revealing the next destination for the message until the last layer of encryption has been pulled of and the message can be delivered to the intended recipient. No individual peer can find out more information than who gave them the message, and to whom it should be passed on. The first node therefore will also know the sender, and the final node in the chain will learn the message. Like with a chain of mixers there is only one trusted node needed in order to prevent the chain from sender to message to be connected (Reed et al., 1998). The largest implementation of Onion Routing is Tor, created by Dingledine et al.. If some attacker can observe the entire network, it can be possible to correlate low latency traffic based on time and bandwidth usage. Therefore, an alternative for Tor could be I2P (`https://geti2p.net`). I2P has better performance for protocols like BitTorrent and implements Garlic Routing (Zantout and Haraty, 2011). Garlic Routing[6] allows peers to bundle multiple messages together, making it harder to do

[6]`https://geti2p.net/en/docs/how/garlic-routing`, accessed 24-07-2017

traffic correlation.

2.4 Peer-to-peer economies

Peer-to-peer networks often are built on communities, and as most communities they are all run by their economies in the end. A result from the Market-Managed Peer-to-Peer Services (MMAPPS[7]) research has been a classification of the different market mechanisms in peer-to-peer networks. Shen et al. incorporate this result in the general description of peer-to-peer market mechanisms in (Shen et al., 2009).

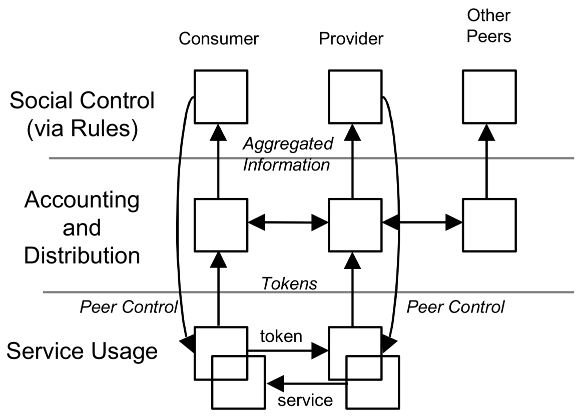

Figure 3: (Strulo et al., 2003), overview image

Antoniadis et al. describe a number of incentives that regulate peer-to-peer file-sharing at the Social Control level in (Antoniadis et al., 2004). They state that in general the peers have rarely an incentive to act in the interest of the entire network, and therefore the efficiency of the network suffers. They go on to describe the incentive structures that could affect the efficiency of the peer-to-peer network. Most of these incentive structures assume that not all the peers will act in the interest of the entire network. Their main finding is that fixed fees or fixed contributions asymptotically attain the same efficiency for small peer groups as if a central regulator with full information would set prices or contributions per peer. In the conclusion of (Antoniadis et al., 2004), Antoniadis et al. suggest that simple incentive systems therefore are preferred for optimal efficiency, to which Courcoubetis and Weber concur in (Courcoubetis and Weber, 2006) about large peer-to-peer systems.

2.4.1 Economical models

Conventional (cloud) data storage providers have an economic model based on the exchange of currency for storage space, bandwidth and proofs of availability. This can be called an **economy of services**, where the providers get payed for the services they provide. Recent peer-to-peer data storage products, such as Storj or Tahoe-LAFS, also use this model but allow payments in a decentralized cryptocurrency.

Peer-to-peer Networks are not just made up from devices and services, but often the backbone of some social structure. Core to file-sharing is the *sharing*, which generally is a community activity. These communities are built by volunteers who not only contribute their data to the community activity, but also time and resources.

This open and volunteer based model is classified as a gift-economy. (Cheal, 1988) Cheal describes a **gift economy** as: “a mode of exchange where valuables are not traded or sold, but rather given without an explicit agreement for immediate or future rewards”. This is not the way companies like Google and Facebook give away services in order to

[7](IST-2001-34201)

create communities. These companies intend to profit from these communities or the secondary products created by the communities. In peer-to-peer communities there often is no acceptable mode for making money, like the refining and reselling of personal data.

Peer-to-peer communities can be closed off or private, and demand some level of participation from a peer to remain part of the community. In many ways this is much like the gift economy. However, in this case the gift is conditional of reciprocity. It is more along the sociology of 'do ut des' - 'a commutative contract whereby something is given so that something may be received in return'[8]. In the extreme case this can become a 'tit-for-tat' exchange - possible giving rise to a prisoner-dilemma - when peers can have negative feedback on each other. The structure is a **reciprocal economy**. An example of a reciprocal economy can be found in BitTorrent networks where download rates are limited for peers that do not contribute.

Finally there is the barter economy following the sociology of 'quid pro quo' in which peers directly exchange services that are mutually wanted. We could call this an **economy of mutual interests**. We find this in protocols like Samsara (Cox and Noble, 2003), that operate with *symmetric contracts.* With symmetric contracts, the services and responsibilities are equal for both peers involved in the contract. Peers also keep each other accountable for upholding the contract, at the threat of breaking both sides of the agreement.

These four economies give a general overview of what to expect from a peer-to-peer network. This makes them useful for discussing the difference between peer-to-peer protocols. However, an economy is always an emergent property of a system and the actual implementation and operation of the protocol. There are not always clear identifiers and sometimes multiple markets and associated economies operate together. For example, possession of a cryptocoin can signify payment for a service, be a badge from some reputation system, or might be an ephemeral intermediary in a symmetrical but asynchronous barter.

2.5 Merkle Trees

There are a number of ways to verify the authenticity and integrity of a file or message using methods from cryptography. The typical way of doing so is using a Hash-based Message Authentication Code, HMAC for short.

An HMAC is a cryptographic way of putting a digital signature on a file, where the signature depends on both the correct digest of a file and the use of the private key of the provider of the file. A digest is a short summary of the file or a identifying number that with extremely high probability belongs to the file. The verifying party can use the public key to check that the integrity of the file has not been changed and actually originates from the original owner.

The most common way to compute a digest is by using a cryptographic hash function. Cryptographic hash functions, like the uniform hash functions[9], map an input of arbitrary length onto a finite set of bins. Aside from the uniform property there are three more properties for a hash function to be a cryptographic hash function: pre-image resistance, second pre-image resistance and collision resistance.

Pre-image resistance is about the difficulty of guessing the input to the hash function from its output. Even if a hash function h is one-way, given message m and a resulting bin $b = h(m)$, b should not leak any information about m. The pocket phone book example, where we create 26 bins from each letter of the alphabet, will not be a good cryptographic hash since it will always *leak* the first letter of the last name.

Second pre-image resistance is the requirement that for a message m_1 and hash function

[8]From Miriam Webster

[9]Hash functions are explained on page 9. (Section 2, subsection 2.2.1)

h, it is difficult to find a new message m_2 for which $h(m_1) = h(m_2)$ holds. This means that it should be very difficult to change a message without the digest changing.

Collision resistance is an even stronger property than the second pre-image resistance. For a hash function h to be collision resistant, it should be difficult to find *any* two messages m_1 and m_2 such that $h(m_1) = h(m_2)$ holds. Hash functions that are collision resistant should make it near impossible to forge messages and makes cryptographic hashes useful for applications where an objects need to be assigned to an 'unique'[10] bin.

Cryptographic hash functions are used in the rest of this paper mostly conceptual, assuming the above properties hold. Whenever an example is required we usually assume hashes from the `SHA`-family. Many of the older references use `SHA-1` (of Standards and Technology, 1995). In 2017, Stevens et al. found a collision in `SHA-1` and it is therefore advised to move to a newer standard such as `SHA-3`. (Bertoni et al., 2011)

Using a file as the input to a cryptographic hash function provides a digest suitable for most applications. This digest should be unique to the file, hard to forge and not leak any information about the file. For the contributions in this project, we use a slightly more advanced structure called the Merkle tree.

The Merkle tree, as patented by Ralph Merkle in 1988, combines cryptographic hash functions with the structure of a binary[11] tree. (Merkle, 1982)

To construct a Merkle tree of height k, divide the data in 2^k equal consecutive pieces called leaves. For each of these leaves you compute the cryptographic hash. These cryptographic hashes are the first branches of the tree. Now proceed to construct the rest of the tree by pairwise combination of two consecutive branches by computing a cryptographic hash of using the results of both previous hashes as input. Compute k layers of the tree until you arrive at a single hash value known as the root of the tree.

A simple Merkle tree with only two leaves will be constructed as follows, using cryptographic hash function h. The first leaves are the hashes of the data m_0 and m_1: $h(m_0)$ and $h(m_1)$. Now we can compute the next (and final) layer by combining the branches of the previous layer pairwise: $h(h(m_0) \| h(m_1))$, where $\|$ denotes concatenation. This is also the root of our Merkle tree.

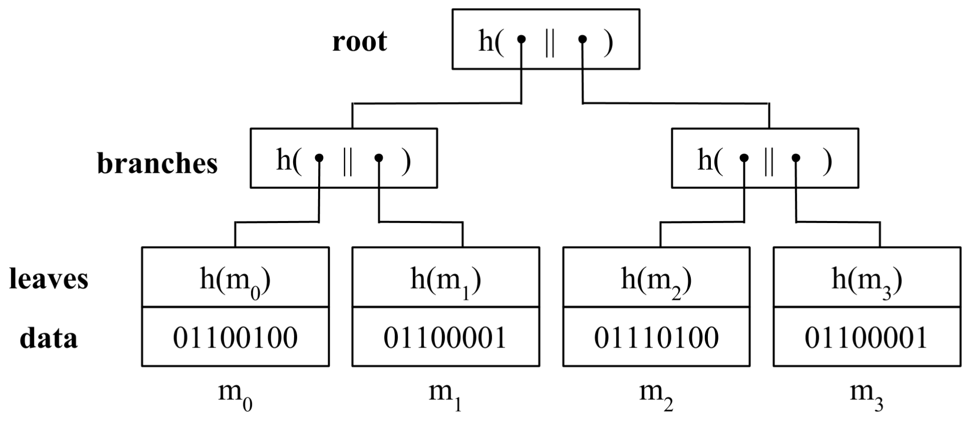

Figure 4: A two layer Merkle tree

To compute a Merkle tree takes $2^{k+1} - 1$ evaluations of the hash function. For a b-bit hash function, storing the tree takes $b \cdot (2^{k+1} - 1)$ bits. It takes $c \cdot 2^k$ bits to store all

[10]If the number of different inputs in the application is smaller than then number of bins the hash functions uses the probability of a collision should move to zero.

[11]We assume a branching factor of two for the examples but any branching factor may be assumed.

the c-bit leaves. Storing the tree therefore takes $\frac{b}{c}(2 - \frac{1}{2^k})$ times the space of storing just the data. In many application the blocks are 32-bit and the hash has an output size of 128-bit, which makes the space needed to store the tree about 8 times bigger for larger k. A trade-off can be made between the computation of a Merkle tree and storing the upper section of the tree in memory. If you save the digest of one of the branches finding a new path in the other branch no longer requires the re-computation of the branch of the stored digest. Saving the upper m branches costs $2^{\log_2 m+1}$ storage spaces, but the evaluations for a single path to a leaf only takes $2^{\log_2 n-m+1}$ evaluations. Given constraints on both storage space and hashing power, Berman et al. prove that there are optimal trade-offs between time and space complexity (Berman et al., 2007).

Once a Merkle tree root is available, you have a digest for the entire file. But this root can also be used to give a Merkle path to a specific leaf of a file. This makes it possible to use the Merkle tree root as a digest for any sub-part of file

A **Merkle path** contains the root, the leaf and k hashes to show that a leaf belongs to the root. The hash on the path at level i is denoted p_i, the Merkle tree root is p_0 , and the leaf p_k. To verify the correctness of a Merkle path, at each level i we use the hash c_i of the adjacent branch. Concatenating the hash on the path p_i with the hash of the adjacent branch c_i gives $p_{i+1} = h(\ c_i \parallel p_i\ )$[12]. For a given Merkle path $(p_0\ ;\ p_k\ ;\ \{c_1, \ldots, c_i, \ldots, c_k\})$ to be correct, the following equation should hold.

$$p_0 = h(\ c_1\ \ldots \parallel h(\ c_i \parallel\ \ldots\ h(\ c_{k-1} \parallel h(\ c_k \parallel p_k))))$$

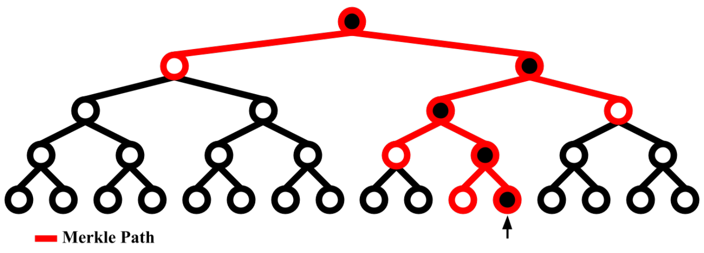

Figure 5: Merkle path

[12]For simplicity we assume concatenation ($\|$) is commutative ($h(A\|B) = h(B\|A)$). When this is not the case, the Merkle path also should contain if the branch came from the left or the right. For the verification the appropriate hash evaluation should be used: $h(p_i\|c_i)$ for left branches and $h(c_i\|p_i)$ for right branches.

3 Requirements

We intend to design a storage product to be used by regular consumers. This storage is for off-site and redundant copies of personal files and data. It therefore ought to compete with external hard drives, Network Attached Storage devices (NAS) and the various cloud storage solutions mentioned in subsection 2.1. After the consumer has bought and configured the product, no interactions should be required to perform its function during the lifetime of the hardware.

There are always multiple parties involved in off-site file storage. As we move away from a model of storage that has a central authority we must consider who else is involved in the process and how much trust is needed in what parties to make the system work. Additionally we must consider the adversaries that present themselves in the absence of a central authority.

The consumers must be ale to trust the provider of the software. This means the software should at least be Open Source such that it can be audited publicly. Even with Open Source Hardware and Open Source Software, as Ken Thompson points out (Thompson, 1984), if the entire product is not completely built from scratch, one must at the very least trust the third parties that provide the fundamental building blocks for our technology. Therefore we must assume that the (hardware) manufacturer of the final product can be used as a trusted party.

With a distributed storage protocol we will have to account for a number of adversaries. The primary adversary in nearly any peer-to-peer network is the free-rider. Free-riders consume (public) resources in the network without contributing anything back. Therefore, Free-riders devalue the rewards honest users get from participating in the network. Another adversary we should consider is the malicious peer. While we can assume that at least half of the peers are honest (altruistic) users, there will always be those act for their own selfish benefits and a minor fraction that would intentionally abuse the system to attack other peers. No other peer should be able to learn which real world identity is related to a particular file stored in the network. Nor should any other peer be able to determine the worth of the stored files. Finally, we consider a global adversary that has the power to observe all the network traffic and wants to learn the identity of a user based on their behavior in the network. The system should be able to protect the user against this adversary.

3.1 Requirements

The objective of this research is to contribute to a peer-to-peer product that is privacy friendly for long term data storage. The product should therefore, in our opinion, at least satisfy the following criteria.

3.1.1 Peer-to-peer data storage protocol

Peer-to-peer networking is the preferred technology in this project to avoid a central authority in the storage protocol. Considering the possibilities and the challenges in peer-to-peer technology (section 2.2), the following requirements have been set.

- The protocol should run fully distributed.
- The protocol should allow a peer to store data in the network.
- The protocol should automatically decide at which other peer the data is stored.
- The protocol should allow a peer to retrieve specific data from the network at a future time.

- The network should - for a (limited) time - retain data if a peer leaves the network.
- The network should be robust against failing or malicious clients.
- After bootstrapping, no parties beyond the (honest) peers should be required to run the protocol.
- At least half of the peers should be honest nodes.

3.1.2 Security Requirements

Security is a general usability requirement of software. Especially when the final product involves (personal) data. (ISO/IEC 27001:2013, 2013) Conditions to safeguard confidentiality, integrity and availability are a necessity, and for our storage protocol have lead to the following requirements.

- The data stored in the network should be (end-to-end) encrypted by the client before storage. (No sharing.)
- The protocol should not leak any information about the data stored or the (private) keys used in this process.
- The network should allow for the integrity of the data to be guaranteed.
- The peers should have a method to verify the integrity of the data stored in the network.
- The availability of the data should be guaranteed over significant periods of time and be robust for perturbations of the network.

3.1.3 Privacy

Privacy is going to be a legal requirement in Europe. Furthermore, privacy is a human right. In this project we interpret privacy as the right to integrity of context (Nissenbaum, 2009).

- The protocol should be privacy preserving: the protocol should leave the users in control of their data and the context of their data.

No data in the protocol should provide information about the actual user, the stored data or the context of the peer to their data. This condition should also hold when there is an adversary of that has the power to observe all the network traffic.

3.1.4 User Friendliness

Security tools and Privacy Enhancing Technologies have have a bad reputation for not being user friendly. Abu-Salma et al. (2017) Seuken et al. have shown that it is possible to hide complex market behavior through the proper design of a good user interface. (Seuken et al., 2010)

However, the protocol should not create any obstacles for creating a user friendly experience.

- The system should work the same way and as simple as storing a file on a local drive.
- The protocol should not require - or even offer - any interaction by the end user with the peer-to-peer network beyond those involved in storage.

3.1.5 Incentives and economies

To make the protocol usable beyond the mere technological capabilities, the following requirements are to help contribute to the viability of implementing the protocol into a product for regular consumers.

- The protocol should offer incentives for good behavior.
- The incentive mechanism should be as simple as possible.
- The protocol should not have a co-dependency with a separate market.
  - Peers should therefore not be allowed to offer currency or (natural)goods in exchange for services on the network.
- If the protocol runs on a gift economy, it should be able to guarantee that the supply of storage will always outrun the demand.

4 A robust protocol for fairness in filesharing

No peer-to-peer protocol has yet been proposed that satisfies all our requirements. So we needed to find reasonable candidate that come close to solving our problem. One approach is to adapt an (anonymous) peer-to-peer file sharing network and include incentives so that it will have a suitably long retention of the data. Alternatively, an existing peer-to-peer data storage protocol can be modified to be *privacy* preserving.

BitTorrent would be an acceptable candidate for the the file-sharing approach. BitTorrent can run fairly anonymous on I2P (Wilson and Bazli, 2016) and has the ability to encourage longer retention. (Menasche et al., 2009) No scheme exists however that lets random peers add files to a BitTorrent swarm and have guarantees for the availability or coverage by the swarm of these new files.

The protocol by Osipkov et al. would be an acceptable candidates for the data storage approach. This protocol is based on robust accounting of the usage and contributions of all the peers in a network without need of a central authority. This gives rise to a reciprocal economy where every peer can consume about as much as they contribute. The protocol attempts to achieve fairness between the peers in the file storage network. Fairness according to Osipkov et al. is that 'peers should be allowed to use at least an amount of network storage comparable to what they provide.' The fairness in their protocol is achieved through a distributed reputation system to keep the network fair for all peers.

In this project we have chosen to pursue the peer-to-peer file storage approach. In the next subsection we describe the reputation system that allows the protocol to achieve fairness. Then we will proceed to detail the messages that make up the Osipkov et al. protocol. In the final part of this section we will discuss some of the shortcomings of the protocol.

4.1 Bartering for file storage

One of the requirements of our protocol is that it should not depend on a separate market. We cannot build an economy of services, since this would require a market of currency. The gift economy would require some external incentive outside of the scope of our technology to require the peers to keep investing in the protocol.[13] It therefore makes sense to consider a bartering economy since it can be regulated through the technology. Fortunately there has been some previous research into peer-to-peer file storage with a bartering economy. The main adversary for this type of economy is the free-rider. Free-riders consume resources in the network without contributing and as such create distrust among the peers in a network.

When regulating a bartering economy we strive for fairness and it is easy to achieve this through a central authority. However, the point of this project is to design a system that does not rely on a centralized authority. An initial idea is to solve this directly in the storage interaction between two peers. When peer A wants to save some data at peer B, peer A immediately has to provide some storage for peer B of the same size. The Samsara protocol (Cox and Noble, 2003) has implemented this. If peer A requests to store data at peer B, in Samsara peer A also has to store an equal amount of data from peer B called a *claim*. The data is only stored if the peer actively can prove that the related *claim* is also stored in the network, otherwise the data is deleted with a set probability p after some time. While the Samsara protocol delivers on fairness, it fails to restore the trust between the peers. It requires constant usage of bandwidth for the peers to keep checking on each other and provides no incentive to keep a file when the other peer goes offline. Having a too high probability p for deletion makes the system unusable for backups. Having a too low probability p for deletion makes the system vulnerable for free-riders. This makes it

[13]Nor do we intend to create a Ponzi scheme.

not useful for the type redundant storage that one needs to restore from when the original device fails.

Osipkov et al. manage to reintroduce an authority to enforce fairness into this system, such that peers can have some trust in each other again (Osipkov et al., 2006). They replace a central authority that could perform this task by the consensus of a small random group of peers. Their protocol uses an Advance Peer Assignment Strategy (APAS), where the public keys of the peers in the system are used to decide on a set of witnesses. These witnesses are other peers in the same network and they are tasked with deciding on the reputation of this peer. This method, as designed by Osipkov et al. does require availability of a *'trusted'* source of entropy in the system to make sure the the set of witnesses is hard to predict for any future point in time, but the current and past sets of witnesses must be easily computable. This allows a storage provider to offload their decision on trusting another peer onto the consensus of the witnesses. Osipkov et al. argue that the network remains functional even if half of the network participants are malicious, given a large enough size of the witness set. They also note that their protocol should not be vulnerable to the Sybil attack, because it assumes a trusted party for the initial key distribution that can verify the identities of the peers.

4.2 The Osipkov et al. protocol

The network consists of nodes (peers) that can offer parts of their local storage space to other nodes in the network. The network has an underlying communication structure based on Chord (Stoica et al., 2001), Pastry (Rowstron and Druschel, 2001), or a similar peer-to-peer Distributed Hash Table, where nodes can find each other and are loosely time synchronized. There is a mutually-trusted authority, that can hand out certificates for new nodes to participate in the network. All nodes have an identity, based on the certificate, that is addressable on the underlying network. At least half of the nodes are interested in participating. Osipkov et al. call this a Community of Common Interest: where at least half of the nodes are interested in having a relatively stable membership in the system for the purpose of consuming and providing long term data storage.

Nodes are responsible for periodically checking the availability of their stored files. It is assumed, though not explicitly required, that nodes encrypt their data before storing it in the network and use erasure codes to segment their files. Erasure codes, such as the Reed-Solomon erasure codes, efficiently encode redundant information into a file for the purpose of error correction and fault recovery. (Reed and Solomon, 1960) With the Reed-Solomon codes it does not matter which bytes of the data are recovered as long as the sum of the sizes of all unique recovered pieces sum to the original file size. Using these erasure codes in part help to hide the file size, and prevent attacks where the storer wants to withhold part of the file for a ransom.

4.3 Protocols and Messages

The storage protocol has algorithms for: queued witness set construction, file storage, file deletion, storage refresh, witness hand-off algorithm, and checking with witnesses. In the next part the operation of the algorithms is illustrated. Pseudocode for all the algorithms can be found in (Osipkov et al., 2006).

Queued Witness Set Construction

Osipkov et al. describe the witness selection process as follows, assuming you have a DHT such as Chord or Pastry to find the user closest to some *id*.

Peer A has an identities based on their public key certificate PKC_A. To find the n-th witness $W_n(A)$ of peer A for time-interval t_c, you select the identity PKC_n that is closest to the hash of $(PKC_A||f(t_c)||g(n))$, for some set functions f and g. Per time-interval t_c the oldest witness is replaced by a new one.

This mechanism reliably and uniformly assigns a number of peers to each and every user. These witnesses are tasked with the accounting of consumption and contribution of the resources in the network. A user will only be served by another peer if the witness group achieves consensus about the current accounting. It is in the self interest of a peer to get a good accounting score. This involves rapid reporting to the witnesses whenever credit is deserved through a contribution to the network.

In practice one can pick the size of the witness-set s_w based on the tolerated fraction of corrupt peers ϵ and the acceptable percentage of witness-sets that have a faulty majority δ.

Let X be a random variable that represents the number of witnesses in a witness set of size s_w that are corrupt. X has a Bernoulli Distribution. We can describe the the fraction δ, the witness set having a faulty majority, as $P(X > \lfloor \frac{s_w}{2} \rfloor)$. The Chernoff bound can be used to calculate a lower bound for the tail of this distribution (Wikipedia, 2017b). $P(X > \lfloor \frac{s_w}{2} \rfloor) \leq e^{-\frac{s_w}{2\epsilon}(\frac{1}{2}-\epsilon)^2}$. Rewriting this equation with respect to s_w gives,

$$s_w = 2\epsilon \frac{\ln \delta^{-1}}{(\frac{1}{2}-\epsilon)^2}$$

as a lower bound for the s_w.

Recovery from a witness-set with faulty majority is generally fast because, assuming at least half of the peers are honest, most faulty majorities are marginal.

Queued witness set W_A construction as presented in (Osipkov et al., 2006).
The witnesses are replaced at random intervals in step 2a. h is a k-bits hash function.

1. On input time t_c (expressed, say, in days) and peer PKC$_A$, initialize $W_{0,\ldots,s_w-1}(A) := \perp, n := 0$.
2. For $j \in \{0,\ldots,e-1\}$ do:
   (a) if $t_c - |t_c| \geq h(j||\mathrm{PKC}_A)/2^k$, set $n := n+1$.
3. For $i \in \{0,\ldots,s_{w-1}\}$, set $W_i(A)$ to the node with `ID` closest successor of $h(\mathrm{PKC}_A \parallel h(\lfloor t_c \rfloor - \lfloor (i+e-n)/e \rfloor) \parallel (i-n) \mod s_w)$.

File Storage

File storage between peer A and peer B begin with a file storage request $S(A)$. Peer A sends a cryptographically signed request to peer B, specifying the identities of both parties involved, the required duration for the storage, the size of the file to be stored and a test for such that peer B can show they actually stored the file. If peer B wishes to accept this file, it informs peer A that they can send over the file, after which A proceeds to send the file to peer B.

Peer B verifies the validity of the sent file based on the provided test in the storage request. If the file is valid peer B produces a receipt $SR(B)$, containing the original storage request, the answer to the test and the current time. This receipt is sent to peer A who can verify its validity.

Peer B sends the receipt to its witnesses set W_B to claim credit for storing files, and also to W_A the witnesses of A. W_B forward the receipt to peer A and W_A, such that W_A can debit peer A for the used storage at peer B.

The test in the Osipkov et al. protocol is to find a pre-image of $h(h(F))$, the double cryptographic hash h over the file F. If the peer B has stored the entire file F, it is easy to compute the pre-image $h(F)$.

File Deletion

All files in the system have an expiration date. Eventually they will be deleted. If peer A wishes to clear debit before that time by deleting the file, the protocol provides a deletion message.

Peer A sends a signed message to peer B, the witnesses of peer A and the witnesses of peer B, that a file belonging to a certain storage receipt can be deleted. The witnesses of peer A and the witnesses of peer B message this request to each other such that the accounting of debit and credit can be updated. All the witnesses will also inform peer B about the deletion, just in case peer B was offline for the deletion request from peer A.

Storage Refresh

If peer A wants to refresh the storage request such that peer B will keep the file beyond the expiration date until a new expiration date, they send a refresh request to peer B containing the original receipt and and updated storage request $S'(A)$ containing the new date. If peer B chooses to honor the request, peer B produces a new receipt $SR'(B)$ and sends this to peer A.

The witnesses are informed analogous to the initial Storage Request. peer B sends the new receipt to their witnesses W_B to claim credit for storing files, and also to W_B the witnesses of peer A. The witnesses in W_B forward the new receipt to peer A and W_A, such that W_A can debit peer A.

Witness Hand-off Algorithm

The witness set of a peer changes over time. As new peers enter the witness set, it is important that they receive the current account, since a single storage request can live over multiple generations of witnesses.

In order to do the hand-off, the new witness requests all the receipts related to storage, storage refresh or deletion from the current witnesses in the set. They then verify the validity of the receipts and keep account of the findings. The witness checks the signatures on the receipts, makes sure the receipts are not expired, and computes if the hash indeed are the pre-images of the requested challenges. If all these checks pass, the witness accepts the receipt for accounting.

Checking with witnesses

If peer A wants to store a file at peer B, peer B first consults the witnesses of A to verify if A indeed contributes as much as they consume. The witnesses then send an assessment based on their accounting to peer B.

4.4 Shortcomings of the Osipkov et al. protocol

With respect to the requirements set in section 3, the Osipkov et al. protocol satisfies most of the criteria. A number of essential requirements are still lacking from the Osipkov et al. protocol.

1. The protocol should be privacy preserving.

2. The protocol should allow a peer to retrieve specific data from the network at a future time.
3. The network should allow for the integrity of the data to be guaranteed.
4. The peers should have a method to verify the integrity of the data stored in the network.

The most critical shortcoming is the fact that the Osipkov et al. protocol is not privacy preserving (1). Even though files are encrypted and erasure codes hide the file size, the network identity of the owner of a file is linked to the storage request. This makes peers traceable based on their non-witness interest in specific files. This also makes it possible to distinguish files in the network from each other based on the owner. If a peer could be compromised through their network identity, a incredibly sophisticated ransomware attack might also try to target their online and off-site backups.

Although there is accounting on the storage of files, and this accounting mechanism incentives fair behavior, the willingness of the peers to offer the files for downloading is absent in this system. This means that there is no protection against denial-of-service attacks for file retrieval. The fairness accounting scheme also cannot help, because the witnesses have no instruments to negatively account the bad behavior. The contracts for which this accounting applies are effectively terminated by virtue of the retrieval of the file.

Both the storage and refresh protocols have a mechanism for checking the integrity of a file at the storer (3), however it is only triggered at the start of the protocol and requires both parties to have the file and do a complete verification. This means that a easy and low latency mechanism to check the availability and integrity of the stored data is missing (4).

5 Contributions

This project has a number of contributions toward the objective of building a privacy friendly peer-to-peer long term storage solution. In section 2 we provide an overview of the relevant literature and technology. We have specified requirements to chose a protocol and in section 4 we argue why the Osipkov et al. protocol is a good starting point towards the objective. This section outlines the two final contributions we add to the project.

In subsection 4.4, we identified a number of shortcomings in the Osipkov et al. protocol. In the next section we propose a method to make the protocol more privacy preserving through the use of anonymous networking. In the last subsection we contribute two methods for the remote querying of file availability. Such tools are beneficial for the trust users can have in the network.

With the project we have contributed to the discussion and understanding of privacy friendly data storage protocols in order to achieve future implementation of a peer-to-peer protocol, based on the previous published protocol by Osipkov et al..

5.1 Anonymity in the Osipkov et al. protocol

The protocol by Osipkov et al. makes very little assumptions about the technology used for networking. In fact, the bootstrapping only requires a Distributed Hash Table and a Certificate Authority to assign each user their identifier.

This is precisely how anonymity networks like I2P work (Zantout and Haraty, 2011) and in more general terms how Tor (Dingledine et al., 2004) organizes its hidden services. Both systems offer an anonymous Distributed Hash Table and a certificate based system for peers to anonymously have themselves contacted through the network. This means that we may assume that peers can be an anonymous part of the network under a certificate-based pseudonymous identity.

In the Osipkov et al. protocol we need an authority to give out the certificates for the peer identities. This is important to protect the protocol against a sybil attack. The identity is also bound to the storage contracts, so it is also important for peers to be able to keep a consistent identity within the network. The pseudonymous identity we assign to the peers has to be authenticated by the certificate authority, consistent over time and not leak information about the real identity. In general pseudonyms are only usable for temporary anonymity. Our challenge will be to keep the pseudonyms unlinkable.

The first step will be to prevent the authority that gives out the certificates to learn the real world identity of the users. We choose for the manufacturer of the devices to be the authority that gives out the certificates, since the users need to trust them anyway not to place a backdoor in the hardware. So the manufacturer puts a certificate in every device. Our applications is in consumer products, so the retailer often breaks the link between the manufacturer and the customer. However, we cannot prevent a curious government or a malicious retailer to read the identity on every device and keep a list of all the customers.

We need a method to put a signed certificate in the devices that does allow the signing of messages using the certificate, but it should be impossible to extract the certificate from the device. This can be done using Physically Unclonable Functions (PUF). A PUF is a physical device that can compute one-way function, is 'inexpensive to fabricate, prohibitively difficult to duplicate, admit no compact mathematical representation, and is intrinsically tamper-resistant' (Pappu et al., 2002). We should note that there has only been two decades of research into PUFs, choosing the right physical system for the task is essential, and implementations still need to consider side-channel attacks. Typical examples of PUF devices are optical systems that can do challenge-response or memory cells that take a set configuration during the installation, such that they can be used to generate a key The latter is useful for our applications. PUFs are at the very least an inexpensive way to increase the cost and time to read the certificate from a device.

In stead of the certificate, the manufacturer can put a PUF in each device. The device creates a public-private key pair using the PUF and asks for a blind signature on the public key from the manufacturer. A blind signature is a valid cryptographic signature on a message, without the signer learning the contents of the message (Chaum, 1983). The device now has a unique, certified public key that can be used as a pseudonymous identity. This certificate cannot be extracted from the device without at least showing signs of tampering. The tampering often breaks the device completely without revealing the key. It should now be very hard to link the pseudonymous identifiers of the peers to their real world identity. These identifiers can be used in the protocol to address peers and witnesses, just like in the normal Osipkov et al. protocol.

For the system to provide anonymity it is remains important for the users to be in an as large as possible anonymity set. Traffic correlation attacks are still the main weakness for low-latency anonymity-networks. If nodes can be fingerprinted or profiled based on their network behavior, and the network behavior is correlated to the behavior on the storage network, then the anonymity set will be reduced to all the nodes with a similar profile. Garlic Routing as used in I2P, can make these kind of correlation attacks more difficult. However even this will not work if the attacker has control over a critical resource.

Consistent behavior of all the peers in the network can increase difficulty of doing a correlation, since it greatly improves the unlinkablity. If larger numbers of peers in the network behave similar, anonymity sets remains large. This is easily facilitated, since most users will run the same hardware and should have no option to configure their device any different than any of the other peers. To an outsider the network should like a large number of indistinguishable peers that push equal sized blobs of high entropy noise around in the network.

Though even Garlic Routing cannot protect the user if they depend on a specific resource. Saving redundant copies in the network, and using Reed-Solomon Erasure Correcting Codes (Reed and Solomon, 1960), as used in (Osipkov et al., 2006) but also in (Wilkinson et al., 2016) or in (Wilcox-O'Hearn and Warner, 2008), allow the user to move on to the next resource if they find one to be unaccessible. This means that an attacker has to be in control of most of the network before attempting a traffic correlation attack.

To achieve regularity, the implementation shall have a number of behavioral conditions, such as the ones listed below.

- Every peer shall attempt to achieve an accounting score of 0. This means offering all available storage to the network and storing as much data in the network. Even if this storages is not yet in active use on the client.
- A peer shall attempt to download every of their stored file once.

The uses of anonymity networking, blindly signed certificates to provide pseudonymous identities and regular behavior should prevent the data of the peers to be correlated to their real world identity. This should help to preserve the privacy of the users, since it allows them to control the context of their data stored in the network.

5.2 Remote Querying

It is important for the trust in a distributed storage network that users have a good sense of the health of the network and the availability of the data they stored in the network. The Osipkov et al. protocol checks at the beginning of a storage contract if the storer actually has the file on disk by demanding a pre-image of the double cryptographically hashed stored file. The availability then is not checked again until the contract is renewed.

A tool has to be provided for the user such that the availability can be verified on a more regular basis.

In the storage literature there are two common Merkle-tree based schemes for verifying the availability of a file using a challenge-response protocol (Wilkinson et al., 2016). The first is to compute a challenge where all the leaves of the Merkle tree are salted with a nonce. Publishing the nonce and the hashed root of the new Merkle tree provides a new challenge. If the storing party still has the data than they can easily find the correct pre-image. An advantage of this scheme is that pre-computation of the challenges allows for querying availability even if the user no longer has the file themselves. The result of the challenge-response can be made public, and is easily verifiable by computing the hash of the response. Shacham and Waters provide a scheme that requires the public key of the challenger to verify the result, making sure that both parties are actually involved in the challenge-response.

Conceptually, the idea of verifying possession of a file is related to proving possession of a private key. That is, if we could equate a file with a private key and derive a much shorter public key, that we give to the verifier, this public key can be used to verify the response of the node storing the file.

There is no simple or common scheme that allows an external party to send the challenge based on a simple digest of the file. There also does not appear to be any cryptographic primitive that allows us to use the entire file as a private key, then compute a *small* public key. The private key can then be used to decrypt any nonce encrypted with the public key to prove the possession of the private key. A further constraint for this problem is that the decryption should obviously not contain a set machine-state before entering the nonce, as this state can easily be stored.

Some initial experiments where the file is considered a large matrix and having the public key be some property of this matrix, such as its eigenvalues and eigenvectors, as a public key were fruitless. In general we found that some other matrix generated by these public properties is also capable of answering the challenge and the file itself does not need to be stored.

An interesting, though not really practical idea is to leverage some cryptography that does allow for a size difference between the public key P and the private key S of a few bits.[14] Using k-bit sized blocks from the file as k-bit private keys and generate m-bit sized public keys ($m \leq k$) allows for a slight compression of $\frac{m}{k}$ of the public digest. This however did only seem to work for toy examples of simple crypto systems and allow the storing user to throw away a file block if it has collision with another public key.

Abandoning the path of the *Proofs*-of-Possession, there are some ways to device probabilistic methods to challenge the availability. We will present two that are based on the computation of Merkle paths. These methods are probabilistic in the sense that not being able to respond allows for the conclusion that the entire file is lost, while any other response gives a probability that the entire file still exists.

5.2.1 Using a counter to create a queriable Merkle tree

For the first scheme we compute the leaves of the Merkle tree with a hashed counter. Let the file F be divided in n blocks denoted $f_1, \ldots, f_n$. We assume that the function $h(x)$ computes the cryptographic hash of x. To construct the Merkle tree we include an index i in the leaves. So the i-th leaf, calculated from the i-th block f_i, will be $h(\ h(f_i) \parallel h(i)\ )$. This construction means that a Merkle path for index i will begin at $h(i)$, uniquely includes $h(f_i)$, and ends at the Merkle root. The scheme therefore is useful for verification by third parties since they can request a path by index. If the root node of the hash tree and the

[14]This idea came from a discussion with Daan Sprenkels.

size of the file (S) are public, any user can send a random challenge $i < S$ to the storer by asking for a Merkle path that starts at leaf i. The ability to produce the Merkle path in a finite amount of time indicates that the storer indeed had the file before, and still has at least a fraction of the data that contains the i-th block of the file and the hashes in the path to f_i. Assuming a user wants to know with a probability p that at least fraction s of the file is available the number of challenges k that need to be answered is expressed as:

$$k = \frac{\log(s)}{\log(p)}$$

Since k trails with p probability of success results in $p^k = s$ probability of surviving a series of k independent trials.

Responses are publicly verifiable but should not be transferable. If just one party checks the possession and the checks are very infrequent, there is an incentive to throw away (part of) the file once the first challenges are over. Complexity and cost of computing with a Merkle tree are discusses in section 2, subsection 2.5.

5.2.2 Size estimation of encrypted files using Merkle trees

An alternative method for testing the availability of a stored file is by estimating its size from statistical properties. Because we deal with encrypted files, the bytes in the data have a uniform distribution. Assuming we have 2^l blocks of l-bits, we expect every possible l-bits sequence to be found as a block. Similar to the previous counter-based method we could send an l-bit nonce to the storer and expect to get a Merkle path in response that begins at a leaf which block is equal to the nonce. There is some reasonable likelihood that a none-answer in this case does not mean that the storer does not posses the file, but that the nonce just was not among the blocks.

We cannot have the storer request new nonces, because then they will just keep requesting nonces until the challenger asks the right challenge. What we can do is send n nonces and assume the fraction that get a Merkle path in the response to be the fraction of the file that the storer possesses.[15] Assuming the queries are all independent we can average the responses and expect that for an honest storer: the standard deviation of N responses decreases with a factor $\sqrt{N}$ and the fraction average asymptotically moves toward 1. A disadvantage of the method is that the storer has to compute the entire Merkle tree for every challenge.

Depending on hardware specifications this might provide reasons to pick one of the methods over the other. The counter method allows to make this trade-off since we just query one leaf per challenge. This is important since we want to make many queries. The performance increase scales exponentially with the size of m. At the other hand, we do value the inspection of the entire file, and specialized GPUs for the parallel computation of the Merkle tree do exists. So if the performance allows for it, the advantages of the file size estimation method could then be exploited.

[15]It is also possible to do size estimation based on other statistical subsections or create more partitions using the Coupon Collector result (Wikipedia, 2017c) such that we should expect near certainty that there should always be at least a single path for a nonce.

6 Conclusion and Future research

The objective of this research was to determine whether it is possible to create a privacy-friendly file-storage product suitable for long-term storage. We set a number of requirements for this protocol based on a study of existing file-storage products and need for privacy-friendliness. The peer-to-peer file-storage protocol by Osipkov et al. was the closest file storage protocol to the requirements. They use a peer assignment scheme to achieve fairness between the peers and prevent free-riding.

However the Osipkov et al. protocol did not yet satisfy all the requirements. It was not inherently privacy preserving, nor did the peers have a efficient tool to monitor the availability of their stored files. We suggested a number of improvements to the protocol such that it can meet more of the requirements.

To become privacy preserving, we propose that the protocol runs on an anonymity network. The protocol only needs a distributed hash table to locate the peers, which works on most anonymity networks. A trusted manufacturer as a certificate authority is needed to for the initial identity distribution. If the implementation enforces regular behavior of all the clients, then using an anonymity network should make the data and metadata unlinkable to the real world identity of the users.

The final improvement was providing the users with better tools to query the availability of their data in the network. Two simple new schemes were proposed to allow third parties to create a challenge to test availability, based on only a Merkle tree root and the file-size. The first hashes a counter into the leaves of the Merkle tree, allowing the creation of challenges based on the counter-index. The second leverages the uniform statistical properties of encrypted files and the ability of the responder to match a specific bit-sequences to do file-size estimation. Depending on the hardware used in the final product and the size the files for which a Merkle tree has to be computed both of these methods have advantages and disadvantages. For large files the performance of the counter method can be several orders better, given some storage and time trade-offs.

While there are a still a number of challenges to be solved, we have shown that it indeed is possible to create a privacy-friendly file-storage product suitable for long-term storage. The Osipkov et al. protocol provides long-term storage in a distributed network. We show that it is possible to make this protocol privacy preserving by means of an anonymity network. Two new tools for monitoring availability should allow consumers to trust the product.

6.1 Future research

All the work in this document is purely theoretical and it will require actual implementation to determine real world feasibility.

The Osipkov et al. protocol has no accounting for the downloading of the files. While it is a fair assumption that honest peers just allow for downloads of the stored file it does exposes the possibility of creating a denial of service by requesting loads of downloads, or for large groups of adversaries in the network to create a bottleneck for a critical resource by remaining honest until a certain target client is in need of a resource. The witness system as it functions now cannot do accounting for downloading, since downloading effectively implies the termination of the contract and the witnesses tally the contracts. An alternative reputation system such as eigentrust (Kamvar et al., 2003) might be better suited to govern the accounting for downloads.

The avenue of solving this problem using a peer-to-peer file-sharing method is still a viable option. This will mostly require motivation peers to also contribute in the sharing of files they do not really care fore. In BitTorrent some progress is made by the application of content-bundling (Menasche et al., 2009). Reputation bases systems to achieve fairness in BitTorrent swarms might be an interesting approach.

Research into the auditing of availability and integrity of remote stored files is often very specific and, just like in this thesis, built under very narrow assumptions. Detecting errors in data stored away at different locations of a data-storage-provider (Spoor and Peddemors, 2010) has a different trust model than doing remote querying on an anonymous peer-to-peer network. Badertscher and Maurer even argue that standard challenge-response audit mechanism, in which the server has to compute a hash $h(F\|c)$ on the file F concatenated with a uniformly random challenge c, is not secure without assuming additional restrictions on the server behavior. An overview of simple and practical methods for doing remote querying of data possession and their security considerations might therefore be a valuable contribution.

Finally it is important to see this work in the scope of popular file storages products. These have often more functionality than just long-term storage of (large) files. Ideally this work becomes just the long term part of a distributed global file-system, that will also allow for privacy friendly file-sharing and anonymous content delivery.

References

Abu-Salma, R., Sasse, M. A., Bonneau, J., Danilova, A., Naiakshina, A., and Smith, M. (2017). Obstacles to the adoption of secure communication tools. In *2017 IEEE Symposium on Security and Privacy. IEEE Computer Society*, pages 137–153.

Agre, P. E. and Rotenberg, M. (1998). *Technology and privacy: The new landscape.* Mit Press.

Antoniadis, P., Courcoubetis, C., and Mason, R. (2004). Comparing economic incentives in peer-to-peer networks. *Computer networks*, 46(1):133–146.

Badertscher, C. and Maurer, U. (2017). Composable and robust outsourced storage. Cryptology ePrint Archive, Report 2017/133. `http://eprint.iacr.org/2017/133`.

Berman, P., Karpinski, M., and Nekrich, Y. (2007). Optimal trade-off for merkle tree traversal. *Theoretical Computer Science*, 372(1):26 – 36.

Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. (2011). The keccak sha-3 submission. *Submission to NIST (Round 3)*, 6(7):16.

Chaum, D. (1983). Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer.

Chaum, D. L. (1981). Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90.

Cheal, D. (1988). *The Gift Economy.* Routledge.

Council of Europe (1950). Convention for the protection of human rights and fundamental freedoms. *Strasbourg: Council of Europe.*

Courcoubetis, C. and Weber, R. (2006). Incentives for large peer-to-peer systems. *IEEE Journal on Selected Areas in Communications*, 24(5):1034–1050.

Cox, L. P. and Noble, B. D. (2003). Samsara: Honor among thieves in peer-to-peer storage. *ACM SIGOPS Operating Systems Review*, 37(5):120–132.

DeCew, J. (2015). Privacy. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy.* Metaphysics Research Lab, Stanford University, spring 2015 edition.

Dingledine, R., Mathewson, N., and Syverson, P. (2004). Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC.

Directive, E. (1995). 95/46/ec of the european parliament and of the council of 24 october 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal of the EC*, 23(6).

EC Reg 2016/679 (2016). Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119:1–88.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382.

Hoepman, J.-H. (2014). Privacy design strategies. In *IFIP International Information Security Conference*, pages 446–459. Springer.

ISO/IEC 27001:2013 (2013). Information technology — Security techniques — Information security management systems — Requirements. Standard, International Organization for Standardization, Geneva, CH.

Kamvar, S. D., Schlosser, M. T., and Garcia-Molina, H. (2003). The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, pages 640–651. ACM.

Lamport, L., Shostak, R., and Pease, M. (1982). The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401.

Lynch, N. (1996). *Distributed Algorithms*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science.

Menasche, D. S., Rocha, A. A., Li, B., Towsley, D., and Venkataramani, A. (2009). Content availability and bundling in swarming systems. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 121–132. ACM.

Merkle, R. (1982). Method of providing digital signatures. US Patent 4,309,569.

Merkle, R. C. (1988). *A Digital Signature Based on a Conventional Encryption Function*, pages 369–378. Springer Berlin Heidelberg, Berlin, Heidelberg.

Miller, A., Juels, A., Shi, E., Parno, B., and Katz, J. (2014). Permacoin: Repurposing bitcoin work for data preservation. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 475–490. IEEE.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

Nissenbaum, H. (2009). *Privacy in context: Technology, policy, and the integrity of social life*. Stanford University Press.

of Standards, N. I. and Technology (1995). Fips 180-1. secure hash standard. *National Institute of Standards and Technology*, 17:45.

Osipkov, I., Wang, P., and Hopper, N. (2006). Robust accounting in decentralized p2p storage systems. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 14–14. IEEE.

Pappu, R., Recht, B., Taylor, J., and Gershenfeld, N. (2002). Physical one-way functions. *Science*, 297(5589):2026–2030.

Park, S., Pietrzak, K., Kwon, A., Alwen, J., Fuchsbauer, G., and Gaži, P. (2015). Spacemint: A cryptocurrency based on proofs of space. Cryptology ePrint Archive, Report 2015/528. `http://eprint.iacr.org/2015/528`.

Pfitzmann, A. and Hansen, M. (2005). Anonymity, unlinkability, unobservability, pseudonymity, and identity management-a consolidated proposal for terminology.

Pouwelse, J. A., Garbacki, P., Wang, J., Bakker, A., Yang, J., Iosup, A., Epema, D. H., Reinders, M., Van Steen, M. R., and Sips, H. J. (2008). Tribler: a social-based peer-to-peer system. *Concurrency and computation: Practice and experience*, 20(2):127–138.

Reed, I. S. and Solomon, G. (1960). Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304.

Reed, M. G., Syverson, P. F., and Goldschlag, D. M. (1998). Anonymous connections and onion routing. *IEEE Journal on Selected areas in Communications*, 16(4):482–494.

Rivest, R. L. and Leiserson, C. E. (1990). *Introduction to algorithms*. McGraw-Hill, Inc.

Rowstron, A. and Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 329–350. Springer.

Seuken, S., Jain, K., Tan, D. S., and Czerwinski, M. (2010). Hidden markets: Ui design for a p2p backup application. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 315–324, New York, NY, USA. ACM.

Shacham, H. and Waters, B. Compact proofs of retrievability. In *Asiacrypt*, volume 5350, pages 90–107. Springer.

Shen, X., Yu, H., Buford, J., and Akon, M. (2009). *Handbook of Peer-to-Peer Networking*. Springer Publishing Company, Incorporated, 1st edition.

Spoor, R. and Peddemors, A. (2010). Cloud storage and peer-to-peer storage.

Stevens, M., Bursztein, E., Karpman, P., Albertini, A., and Markov, Y. (2017). The first collision for full sha-1. *IACR Cryptology ePrint Archive*, 2017:190.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160.

Strulo, B., Smith, A., and Farr, J. (2003). An architecture for peer-to-peer economies. In *Peer-to-Peer Computing, 2003.(P2P 2003). Proceedings. Third International Conference on*, pages 208–209. IEEE.

Sweeney, L. (2002). k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570.

Thompson, K. (1984). Reflections on trusting trust. *Communications of the ACM*, 27(8):761–763.

Travers, J. and Milgram, S. (1967). The small world problem. *Phychology Today*, 1:61–67.

Van Den Hooff, J., Lazar, D., Zaharia, M., and Zeldovich, N. (2015). Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152. ACM.

Warren, S. D. and Brandeis, L. D. (1890). The right to privacy. *Harvard law review*, pages 193–220.

Westin, A. F. (1967). Privacy and freedom, atheneum. *New York*, 7.

Wikipedia (2017a). Burstcoin — wikipedia, the free encyclopedia. [Online; accessed 24-July-2017].

Wikipedia (2017b). Chernoff bound — wikipedia, the free encyclopedia. [Online; accessed 17-August-2017].

Wikipedia (2017c). Coupon collector's problem — wikipedia, the free encyclopedia. [Online; accessed 26-July-2017].

Wikipedia (2017d). Napster — wikipedia, the free encyclopedia. [Online; accessed 15-August-2017].

Wilcox-O'Hearn, Z. and Warner, B. (2008). Tahoe: the least-authority filesystem. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 21–26. ACM.

Wilkinson, S., Boshevski, T., Brandoff, J., and Buterin, V. (2016). Storj a peer-to-peer cloud storage network.

Wilson, M. and Bazli, B. (2016). Forensic analysis of i2p activities. In *Automation and Computing (ICAC), 2016 22nd International Conference on*, pages 529–534. IEEE.

Zantout, B. and Haraty, R. (2011). I2p data communication system. In *Proceedings of ICN 2011, The Tenth International Conference on Networks*.