AALBORG UNIVERSITY

STUDENT REPORT

Decentralised and Trustless User-Driven Rating Platform Resilient to Attacks

Project Group:
DEIS101F18

Supervisor:
Brian Nielsen

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Selma Lagerlöfs Vej 300
DK-9220 Aalborg Ø
http://www.cs.aau.dk

Title:
Decentralised and Trustless User-Driven Rating Platform Resilient to Attacks

Theme:
Distributed Systems

Project period:
Spring semester 2018

Project group:
deis101f18

Author:
Mathias Vestergaard Rasmussen

Supervisor:
Brian Nielsen

Pages: 46

Date of completion:
June 15, 2018

Abstract:
A majority of information on the internet is served and accessed through a relatively few number of service providers. This structure gives service providers a large influence over how content is accessed. In this report, we consider a decentralised and user-driven rating platform, independent of a trusted authority, where users controls the availability of rating data. The platform enables applications and users to integrate a rating system, and provides methods for assessing the relevance of content. We introduce challenges associated with developing a decentralised and trustless system that manages rating data, and considers possible methods and technologies that can be used to solve them. We argue that the primary thread to the system are Sybil attacks, and we propose a method to mitigate attacks against the rating system, based on a proof-of-work system. Based on an evaluation of attack resilience, we argue that the system is feasible to implement and operate, but requires a combination of different types of technology that is not straightforward to implement. We conclude that a complete implementation is necessary to evaluate the effectiveness of the system.

Reading Guide

The content of this report is intended to be read in order of appearance. Chapters can be read by themselves, but arguments may rely on past chapters. All figures in the report are made by the authors unless stated otherwise. Component, class and sequence diagrams are based on the Unified Modeling Language (UML)[1] Specification Version 2.

Citation Style

All references throughout the report are in IEEE style. The bibliography can be found at the end of this report on page 49.

[1] https://www.omg.org/spec/UML

Summary

In this report we have considered a decentralised and user-driven rating platform, that enables applications and users to integrate a rating system, and provides methods for assessing the relevance of content. Content assessment is needed by services to be support aspects, such as filtering out irrelevant content and ranking content according to popularity or relevance. We examine methods for making the system trustless, in terms of making it independent of a trusted authority.

The motivation for a decentralised and trustless rating platform driven by users, is based on providing an alternative to typical rating systems, which is controlled exclusively by a single service provider. This structure makes service providers responsible for how content and ratings are accessed. With a common and open rating platform, rating data can be shared between services, and be controlled by users.

We examine different types of rating systems, and consider used-based collaborative filtering for the proposed system.

In section 2.3, we examine potential threads and attacks against the system, especially considering the Sybil attack. In section 2.4 we present different methods and technologies that can be used to avoid and mitigate the examined threads against the system, with proof-of-work as the primary approach against Sybil attacks. We also examine cryptographic hash functions and digital signature, which is considered as a core part of the proposed design, to ensure data integrity and authenticity, which are important aspects in our effort to make the system trustless.

The design of the system, presented in chapter 3, is based on three essential data types: user, metadata and review. A user represents a user identity in the system and is based on a digital signature system. Metadata is associated with arbitrary types of content, and provides an abstraction for describing and identifying content, external to the system. A review represent a rating of metadata and associated content, unique to specific user. Reviews are signed using digital signature to ensure authenticity, and appends proof-of-work data, to make it computationally expensive to produce a large number of reviews. This serves to mitigate attacks, where an attacker relies on adding many reviews to subvert the rating system, and manipulate the final predictions. The system architecture is based on a number of components with responsibilities divided into: system management, data integrity and authenticity, proof-of-work, collaborative filtering, storage management, and communication.

In chapter 4, we examine different existing technologies that can be used to implement the different components of the system.

We evaluate the proposed system, in chapter 5, according to scalability of disk space usage, and attack resilience in terms of attacks to different methods of collaborative filtering, and the effect of a proof-of-work approach to mitigate the attacks.

Contents

1 Introduction

1.1 Motivation

Today most internet services is shaped around an ecosystem of large corporate controlled platforms [1]. This means that a majority of information is served and accessed through a relatively few number of service providers. Web services such as Google, Facebook and Twitter has made it user friendly for people to publish and access information. However, the structure of these platforms gives single entities enormous influence over what content the public consumes, and control of what can be accessed and published. The platforms controls aspects such as censorship and moderation of content.

Companies are driven by monetary incentives to sustain these platforms. An example of this is a prioritisation of viral content, rather than the interest of users, to increase advertisement revenue.

At the same time, decentralised applications and services are emerging, in the form of file and data sharing, cryptocurrencies and the decentralised web. BitTorrent is an example of a decentralised file sharing network, that has seen wide adoption, with 15–27 million active users online [2]. Recently cryptocurrencies and blockchain technology, popularised by Bitcoin, have been subject to research and development in a wide number of areas [3]. The decentralised web has recently seen development, in the form of projects such as the Interplanetary File System (IPFS). A motivation for decentralised services is to shift publishing and discovery into the hands of users instead of companies, and without intermediaries, censorship can be avoided [1].

A common necessity for services that manages content, is the ability to assess it in terms of e.g. quality or relevance. This serves purposes such as providing content filtering, ranking and recommendation. Large platforms often utilises a rating system to perform content assessment, by relying on users to rate content, and use this information to improve how content is accessed and presented.

In a decentralised system, without a trusted authority, it is difficult for a user to judge if content is relevant and authentic. A common and open rating platform with focus on decentralisation, could make it possible for users to provide feedback and retrieve information to assess the relevance of content, across multiple applications, based on communities of users. The rating platform could provide large amounts of rating data, from a broad range of areas. The platform would be operated by users, and a desire to acquire relevant content, instead of a single organisation.

1.2 Project Description

In this report, we consider how to assist users with determining the relevance of content, based on common rating platform. The platform is decentralised and driven by users, and

allows applications to interact with it to provide rating services and content assessment.

A movie based service is an example of an application that could use this system. Users would rate movies, and based on ratings from other users with similar opinion, predictions of how a user would rate other movies could e.g. be used to make recommendations. The rating data could be shared with other similar services, such as a site for rating books, to improve the initial user experience, by avoiding the need for users to rate a number of similar items before getting appropriate feedback.

Another example is to use ratings for judging the authenticity of content, such as files. An application could make users that rate content in terms of verifying whether it is authentic or not. The rating data could then be used to filter out content that are likely to be irrelevant to users, such as fake content and spam. If the same or similar content, such as a file, is used in multiple services that integrates the platform, the file could be assessed based on ratings from users that does not necessarily use both services themselves. If the file contains spam or a virus and receives a bad rating in one application, it could be filtered out in other services as well.

The basic functionality of the system is outlined in figure 1.1. Users should be able to insert reviews for specific content, and be able to retrieve predictions for the relevance of content to a given user.

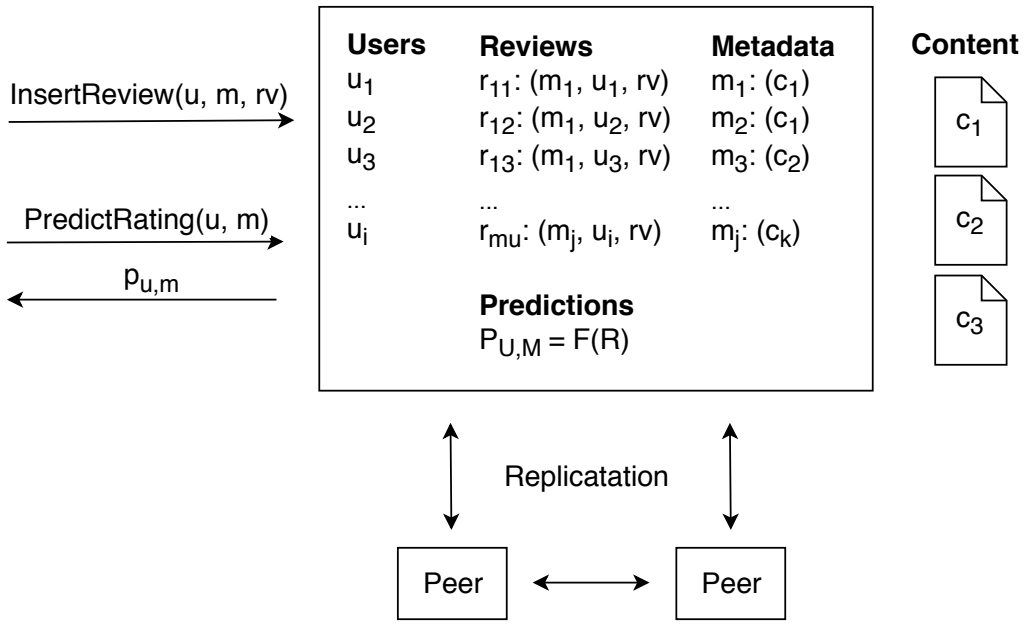

Figure 1.1: Overview of basic rating system functionality

The system consists of a set of user identities U, a set of content metadata M, and a set of reviews R. Metadata is used to describe and identify specific content, such as a movie or file. Each review r represents a rating value rv and is uniquely associated with a user u, and content metadata m, as defined by the following set:

$$R \coloneqq \{ (u,\ m,\ rv) \mid u \in U,\ c \in C,\ rv \in \mathbb{R} \}$$

Based on the set of reviews, an estimate of the relevance of content for a specific user can be predicted.

In a decentralised environment, multiple peers are operating the system, and can have different subsets of the total set of data, as defined by the set of peers P:

$$P \coloneqq \{(U_p,\ M_p,\ R_p) \mid U_P \subset U,\ M_P \subset M,\ R_P \subset R\}$$

We distinguish between user and peer. A user is an identity in the system. The identity is created by a person and used to perform actions in the system, such as creating a review. A peer is an actively running instance of the system. A peer is often controlled by a person who is also a user of the system.

Over time peers will update their sets of data, by communicating with other peers, as data such as new reviews are added by users.

2 Analysis

This chapter will present an analysis of the problem domain.

2.1 Problem definition

In this section we consider concepts and terminology used to describe problems related to this project.

2.1.1 Content Assessment

Many web services and applications allow users to submit content. In order for service providers to administer user submitted content, the systems usually integrates access control, relying on an authoritative server to create and verify user accounts allowed to interact with the service. This also allows the service to monitor the activity of users.

In cases where users cannot be assumed to adhere to the rules and guidelines of a service, it is not sufficient to constrain who can use the service. Instead a moderation system can be used, where people are assigned as moderators, to review user submitted content, and decide whether to accept or reject it.

Content assessment can also be distributed by allowing an existing community of users to express their opinion, rather than having few individuals moderate it alone. This can be achieved by using a rating system that allows users to rate content, e.g. based on a five-star rating scale or a like/dislike feature. Content ratings allows the service to filter or rank content according to its popularity or usefulness.

Without performing content assessment, services with a large number of users could be overflowed with excessive amounts of inappropriate or irrelevant content.

2.1.2 Challenges with Decentralisation

Content assessment in a decentralised environment involves several challenges.

The type of content moderation, described in the previous section, relies on an authority that is in charge of authorising people by giving them special privileges, such as moderation. In systems with an authoritative server, user accounts can be registered and it is possible to verify the identity of a user, more or less reliably, using external information, such as an email address.

However, in a decentralised system where anyone is allowed to join and operate the system, no entity can be inherently trusted to perform verification. Adding an authority to the system would make it dependent on a single point of failure, and make it susceptible to censorship. Similarly, using a typical rating system together with a decentralised service that manages content, undermines the potential of the service, by introducing a dependency that could become unavailable to users. With a trusted authority, a service

can monitor user activity and react upon inappropriate usage, however without an authority, no single entity can make decisions on behalf others on how to react.

Trustless System

A decentralised system that does not depend on an authority, can be said to be trustless, in the sense that users are not required to trust any specific entities, in order for the system to be functional. However, users usually have to establish trust between each other to take advantage of the system.

Without a trusted authority, problems arise that would otherwise have been trivial to solve. These problems must be tackled differently. This involves defining how users are to establish a notion of trust between each other, and choosing an approach for how to manage information that could have otherwise been assumed to be trustworthy.

In [4], two definitions of trust is considered: reliability trust and decision trust. We consider trust according to the definition of decision trust:

> *Trust is the extent to which one party is willing to depend on something or somebody in a given situation with a feeling of relative security, even though negative consequences are possible. [4]*

This definition is somehow vague, but captures the idea of representing a measure of trust between users, that does not guarantee a specific outcome. A measure of trust between users is intended to improve the system and the user experience, however inappropriate assignment of trust could have the opposite effect.

Explicit Trust

An approach to solve some of the issues in a trustless system, is to establish an explicit network of trust relationships between users. This can work well for applications such as social networks, where users can use a subscription model to explicitly choose who to trust, e.g. who to follow or become friends with. Such a network of trust can be used to determine how content is shared between users. However for applications, where content is intended to be shared publicly, relying on an explicit network of trust could be a limiting factor for allowing other interested users to explore and access it. Maintaining an explicit and useful trust relationship between users can be difficult and time consuming, since it is not always apparent, in all services, which users that are trustworthy.

To avoid explicit assignment of trust, we will examine how rating systems might be used to derive a notion of trust between users.

2.2 Rating Systems

In this section we examine different type of rating systems, based on ratings of user transactions, and ratings of content.

2.2.1 Reputation System

Reputation systems provide a service for users to build and assess trust.

Reputation is based on how users rate transactions between each other, and can be considered as a collective measure of trustworthiness [4], based on ratings from other users. Reputation systems considers aspects independent of opinion, such as quality of a service or item, and assumes that users will judge transactions consistently.

Examples of existing services that integrates reputation systems, by allowing users to rate transactions between each other are:

- eBay: an online auction site that allows buyers and sellers to give feedback for each transaction, in the form of a rating (positive, negative or neutral).
- Stack Exchange: a network of question-and-answer sites where users can rate questions and answers.
- Trustpilot: a site that lists the reputation of online businesses based on user reviews.

These systems uses ratings to derive reputation for users, which can be used to guide decisions about future interactions, as well as moderate how content is presented to the user. This can, for example, be in the form of considering the rating of a transaction with a user, or the rating of content authored by a given user.

2.2.2 Collaborative Filtering

Collaborative filtering is an approach used in recommender systems [5]. It is based on collecting and analysing information on user behaviour and preferences, typically based on how users rates different items.

User-based collaborative filtering considers the similarity of users subjective opinion. It relies on the assumption that users will rate content differently based on subjective taste, and that users with similar taste will rate content similarly.

By considering the similarity between how users rate certain items, the method can be used to make predictions for how a user would rate given items. This information is typically used to recommend a list of items to a user, based on items with a high predicted rating. Since the approach can be used to predict how a user would rate any individual item, it can serve the purpose of estimating how likely a user is to be interested in an item or find it inappropriate.

Examples of services that uses collaborative filtering, by allowing users to rate content such as comments, movies and articles are media such as Facebook, Netflix, Reddit and YouTube.

Collaborative filtering can be grouped into memory based and model based techniques. Memory based techniques are effective and easy to implement, but performance decreases with sparse data and does not scale well. Model based techniques uses machine learning algorithms and typically handles data sparsity, through dimensionality reduction. An issue related to data sparsity is the cold start problem. It concerns that accurate predictions cannot be made until a sufficient amount of information has been gathered, which

requires users to rate a sufficient number of items. By creating a platform that multiple applications can utilise, it might be possible to minimise the problem by reusing ratings across applications.

2.2.3 Comparison

Collaborative filtering systems are similar to reputation systems in that they both collect ratings from users. A rating in collaborative filtering represents a users opinion about specific content, while a rating in reputation systems represents the quality of a transaction a user has had with another user.

Collaborative filtering [4] provides subjective scores for each user, by making calculations based on opinions from communities of similar users, whereas reputation systems provides an objective reputation score of each user, by performing a global calculation based on all transactions.

Collaborative filtering assumes that users can be trusted to provide their genuine opinion, as opposed to reputation systems, that assumes users might attempt to misrepresent the quality of transactions, such as giving low ratings to competitors.

Some services combines reputation systems and collaborative filtering. Recommendations can be improved by considering reputation, or aspects such as the similarity between users can be used to determine reputation.

2.3 Threads and Attacks

In this section we examine threads to the system, in terms of attacks that can be performed to disrupt the operation of the system.

2.3.1 Denial-of-Service Attack

An attacker could initiate a denial-of-service (DoS) attack by e.g. establishing many connections to peers and cause transactions of poor quality, such as sending and requesting excessive amounts of data, and limiting the speed and bandwidth of data transmission to delay the time for handling each request.

2.3.2 Inauthentic Reviews and Content

An attacker could attempt to act on behalf of another user, and insert inauthentic reviews into the system, i.e. reviews that have not been created by the claimed user. This is known as masquerading or impersonation. Masquerade attacks can be avoided by verifying that reviews are actually created by the claimed user. Methods for ensuring review authenticity will be examined in section 2.4.2. However, given that peers have to verify reviews, an attacker might cause a DoS attack by sending a large number of invalid ratings, to overwhelm peers with checking them.

An attacker could also attempt to send false data, potentially containing a virus, when a user is requesting and expecting something else. To prevent this, the integrity of data

must be ensured, which will be further examined in section 2.4.1. However, similar to inauthentic reviews, peers could still be subject to DoS attacks, if they need to check the data for each request.

A reputation system can be used to reduce how often peers interact with attackers. This can provide resilience to some types of attacks, e.g. a user can avoid checking invalid data from an attacker. It involves making peers share the quality of their transactions with other peers. Peers that sends invalid reviews or data, would get a low reputation score, which would punish attackers, and encourage other peers to avoid transacting with them. Examples of using reputation systems in peer-to-peer networks will be examined in section 2.4.3.

A problem with the approaches mentioned above, is that they do not account for the possibility of an attacker creating multiple user identities. In that case it is possible to insert valid reviews from multiple user accounts, and also to manipulate the reputation score. This is known as a Sybil attack.

2.3.3 Sybil Attack

In a decentralised environment with potentially unknown remote peers and no trusted authority, it cannot be determined if a single entity controls multiple identities in the system [6]. Without a trusted authority to validate the correspondence between entity and identity, an attacker could attempt to carry out a Sybil attack.

In a Sybil attack, an attacker creates many pseudonymous identities to gain larger influence in a system. By controlling a large number of peers in a peer-to-peer network with a reputation system, the reputation score of users can manipulated. The attacker could utilise different strategies, such as making the pseudonymous users provide positive feedback to each other, while giving other users negative feedback, in order to increase the relative reputation score of pseudonymous users.

Effects of the Sybil attack depends on the cost of creating identities, and whether or not different identities have equal influence in the system. E.g. new identities can be given less influence, however an attacker can mitigate this by creating the identities beforehand, prior to the attack.

If the trust relationship between users are be established explicitly, as mentioned in section 2.1.2, the effects of Sybil attacks are extremely limited, because the attacker cannot affect reputation or trust for other users, and it is unlikely that users would choose to trust more than a few of the pseudonymous identities, if any at all.

Shilling Attack

Similar to reputation systems, collaborative filtering is vulnerable to the insertion of pseudonymous identities. In this case, the attack is usually referred to as profile injection or shilling attack, and the identities as attack profiles [7]. In the attack, ratings are inserted into the system to increase or decrease the overall rating of specific items, in order to affect the final recommendations.

An average attack is an effective type of attack, where the attack profiles establishes similarity with users by rating randomly chosen filler items. The ratings are based on knowledge about the mean of each item.

Items can be rated to establish high similarity with a specific community of users, in order to affect their recommendations. By rating popular items, it is possible to get a high correlation with a large proportion of the userbase, which can be used to cause an effective attack.

2.4 Solution Elements

In this section we examine existing methods to mitigate the threads and attack scenarios described in the previous section.

2.4.1 Cryptographic Hash Function

To verify content integrity and ensure that data has not been corrupted during transfer, hash-based verification can be used to check that received data corresponds to the requested data.

A hash function takes an arbitrary message as input and produces a fixed size value, known as the message digest or hash. A message can be checked by comparing its digest with a digest known to be correct.

In order to secure the system against the possibility for attackers to tamper with data, and send inauthentic or false content, as mentioned in section 2.3.2, a classical hash function is not sufficient. If an attacker can cause a hash collision, i.e. construct a message that produces the same digest as an authentic message, a forged message would not be detected.

Cryptographic hash functions [8] are a special class of hash functions. They tend to be more computationally expensive, but have certain properties, required to ensure the integrity of content and protect against message forgery. This includes collision resistance, meaning that it is unfeasible to find messages with the same digest.

By utilising a cryptographic hash function, and allowing users to obtain and refer content by its digest, data can be secured against corruption and tampering.

Content-addressable storage

Metadata associated with a specific review, must be referenced by the review to make it possible for peers to acquire it.

Content-addressable storage (CAS) allows the system to refer to metadata, without needing to know its exact location beforehand.

Typically content-addressable storage refers to content by its digest [9], generated with a cryptographic hash function. The digest serves as a content address, which is permanently fixed to the content, making it impossible to change the content without also changing the content address. For this reason it is often not permitted to edit information

once it is stored in a CAS system. This is suitable for the system, since it should not be possible to change metadata that is referenced by a review.

2.4.2 Digital Signature

Digital signature [8] is a technique that can be used to demonstrate the authenticity of a digital message. While a cryptographic hash function can be used to ensure data integrity, digital signature can be used to ensure that data, such as a review, has been created by a specific identity.

Both schemes ensures data integrity, because a message cannot be altered without also changing and invalidating the corresponding digest or signature. Digital signature algorithms usually produces a signature based on a message digest and not the message itself. This is considered sufficiently secure, given the properties of cryptographic hash functions.

Digital signature is based on public key cryptography, and involves algorithms used for key pair generation, signing messages, and verifying signatures:

- An identity is represented by generating a private key and a corresponding public key.
- A signing algorithm uses a private key and a message to produce a signature.
- A signature verification algorithm takes a signature and a public key, and checks that the result corresponds to a given message, in which case the message is considered authentic to the identity associated with the public key.

Some essential properties provided by digital signature systems are: message authenticity, data integrity, and non-repudiation of origin. Message authenticity can be used to ensure that reviews originates from the same source, whether it is trusted or not. Data integrity ensures that data cannot be tampered with. Non-repudiation of origin is similar to authenticity, but considers the aspect that an entity cannot later deny having signed a message. This property can be used to ensure that reviews cannot be easily removed by the author, e.g. to avoid attempts by an attacker to later become disassociated with false data.

These properties only holds given that the private key of a user has not been compromised. To reduce effects of a compromised private key, the system could allow users to announce revocation of a key-pair, which indicates that the keys can no longer be considered valid.

Because it is unfeasible to produce a valid signature without the corresponding private key, digital signature can ensure that attempts to insert inauthentic reviews into the system is detected.

2.4.3 Reputation Management in Peer-to-Peer Networks

Countermeasures to attacks against peer-to-peer networks, such as distribution of inauthentic and false content, as mentioned in section 2.3.2, has been the subject of

previous research. We examine different methods that have been proposed for managing reputation in peer-to-peer networks.

EigenTrust [10] is a popular reputation management algorithm for peer-to-peer networks. The algorithm assigns a global trust value to each peer in the network, based on their history of satisfactory and unsatisfactory transactions with other peers. It works by making each peer compute local normalised trust scores for all peers. A global trust value, or reputation score, is computed by aggregating the local trust scores using power iteration. This assumes a notion of transitive trust, in the sense that trust is propagated such that a peer relies on the judgement of a trusted peer's opinion of other peers. Peers can evaluate which peers to interact with, by using the reputation scores as an indication of the likelihood that another peer will participate in a satisfactory transaction.

EigenTrust++ [11] is an extension of the EigenTrust algorithm aiming to be more resistant to attacks. The algorithm also integrates techniques based on another reputation algorithm called PeerTrust [12], such as a credibility measure. It works by computing a feedback credibility score, based on the similarity between peer feedback, such that peers with more inconsistent feedback is considered less credible. When computing the global trust score, the feedback credibility is used to weight the aggregation of local trust values. Instead of the trust propagation used in EigenTrust, which was based on a uniform distribution, EigenTrust++ uses a threshold-based probabilistic trust propagation, by considering a weighted combination of local trust and feedback similarity, to cut off propagation of trust to malicious peers. This blocks propagation of positive ratings to malicious peers and negative ratings from malicious peers.

Reputation systems can be used in peer-to-peer systems to derive reputation-based trust, established in terms of quality of service provided by peers. Transaction can be judged according to aspects such as the authenticity of received data, network speed, etc. While some systems use advanced methods to manage different scenarios of malicious peers, reputation systems are still vulnerable to Sybil attacks, in case an attacker manages enough identities to gain significant control of the final reputation scores.

2.4.4 Robust Collaborative Filtering

Robust collaborative filtering [13] algorithms aims to make collaborative filtering more robust against attempts to manipulate ratings and recommendations, such as the shilling attack mentioned in section 2.3.3.

Typically, robust collaborative filtering algorithms performs a detection of attack profiles, and the performs regular collaborative filtering on subset of the data, in which the attack profiles have been removed. An example of an algorithm that can be used for attack profile detection is VarSelect [13]. It exploits that attack profiles, that works together in groups to maximise the effect of an attack, tend to be highly correlated. The algorithm uses principal component analysis (PCA) to determine which users adds least information. VarSelect can be used efficiently together with collaborative filtering algorithms based on singular singular-value decomposition (SVD), because SVD and PCA a related and can share computational steps.

The performance of robust collaborative filtering algorithms depends on the type and

size of the attack. In the system that we consider, rating information is publicly accessible. This means that attackers can perform and informed attack, i.e. apply strategies that are more effective on the existing rating data and algorithm used for predictions. We still consider robust collaborative filtering advantageous to conventional collaborative filtering, because attacks are more difficult to perform and obfuscation of an attack, to avoid detection, can decrease its effect. However, because effective attacks are still possible, we do not consider robust collaborative filtering alone, sufficient to mitigate potential attacks against the system.

2.4.5 Proof-of-Work System

Proof-of-work is a technique used to mitigate denial-of-service and spam attacks, and was first introduced in [14]. In a proof-of-work system a client must solve a problem, known as a client puzzle, before an action, such as a service request is considered valid and will be handled. A client puzzle usually consists of a mathematical problem, solved through computational work. Proof-of-work systems assumes the asymmetry of proof-of-work data being time-consuming to produce and fast to verify.

The intention is that ordinary users should not be significantly inconvenienced by the processing time required to perform an operation, whereas the impact on attackers, who needs to perform a large number of operations, will make the attack unprofitable. As an example, to mitigate the effects of a Sybil attack, described in section 2.3.3, a proof-of-work system could be used to ideally make it unprofitable for an attacker to operate a large number of identities, given the amount of time or computational resources required to execute a successful attack.

Hashcash [15] is an example of a popular proof-of-work system, originally used to reduce email spam, and more recently as part of the mining algorithm in Bitcoin. The computational work consists of continuously searching for a partial preimage of hash function, i.e. finding a message that produces a specific digest. In practice, it searches until a hash value is found, where all the initial bits equals zero. The needed number of initial bits defines the difficulty of the task. A service provider can efficiently hash the found message and verify its validity.

Memory-Bound Proof-of-Work

A problem with proof-of-work systems, such as Hashcash, is that as computers become faster the difficulty of the computation should be increased. However, because the relative speed of CPUs can differ significantly between peers, it might still be feasible for people with access to advanced hardware to successfully perform an attack. With the development of dedicated hardware to compute proof-of-work with increased throughput, mainly motivated by mining of cryptocurrencies, alternative schemes such as memory-bound functions has become of particular interest.

In contrast to compute-bound functions, where computation time is determined by the speed of the processing unit, the computation time of memory-bound functions are dominated by the time spent accessing memory. The speed of CPUs has increased at

a significantly higher rate than the speed of main memory [16]. This means that the difference in performance between CPUs, is often greater compared to the speed of main memory.

Memory-bound proof-of-work systems can be considered as an egalitarian approach [17], because the performance gap between hardware of a typical user and an attacker, is significantly lower compared to compute-bound methods. Utilisation of memory-bound systems that requires intensive use of RAM in terms of both size and bandwidth, makes the cost of finding a solution comparable across different processing units, such as CPU, GPU, FPGA, and ASIC. A desirable characteristic for these proof-of-work systems, is to have a steep time-memory trade-off. This means that to increase the computation speed, the use of more memory is required, which makes it inefficient to parallelise the algorithm.

To reduce the amount of potential Sybil attacks and mitigate its effect, a proof-of-work system can be used to make it costly or time-consuming for an attacker to execute an attack, in terms of creating valid user accounts and reviews.

2.4.6 Communication Protocol

For the system to be decentralised and trustless, and allow peers to individually operate the system, we consider possible communication protocols that can enable peers to communicate and synchronise or replicate data in the system, such as reviews and metadata.

Distributed Hash Table

A distributed hash table (DHT) can be used to distribute data across many peers, and provides structured network overlay that can be used to store and lookup data. DHTs are a highly scalable, a popular example is the BitTorrent Mainline DHT with millions of active users [2].

DHTs might provide a useful way for peers to store and access data, but because peers typically needs large amounts of the rating data to compute proper predictions, they might already store the rating data and not require it to be distributed. Another case is that peers primarily stores ratings from the most similar users, and in that case using a DHT might be inappropriate because peers does not control how the data is distributed.

Gossip Protocol

Gossip or epidemics protocols [18] is another method that can be used to implement a large scalable network of peers. They can be efficient because they avoid the overhead of connecting or broadcasting information to every peer in a network.

Gossip membership protocol usually form an unstructured network overlay, in contrast to a structured one, such as DHTs. This means that peers are not connected according to a fixed network structure.

We have examined the different gossip membership protocols SCAMP [19], CYCLON [20] and HyParView [21], which all uses a partial view of the network, meaning that they do not have a complete list of all peers in the system. This makes them highly scalable and efficient when a large amount of peers are active. The protocols are highly dynamic and fault tolerant.

2.5 Delimitation

In this section we delimit the problem area that we will be focusing on in the remaining part of the report.

2.5.1 Rating Systems

Both reputation systems and recommender systems provides techniques for evaluating the relevance of content, but focuses on different aspects. Reputation systems can be used to derive reputation or trustworthiness of users. This can be used to judge content based on the reputation of the users that are distributing it, however the quality of a transaction may not be related to the quality of the content itself. Defining a notion of trust between users, based on reputation systems, is straightforward when using the reputation score.

User-based collaborative filtering can be used to predict what content a user might find interesting, based on ratings from other users. This has the benefit of considering the quality or relevance of content itself, which provides a more straightforward way of assessment. Compared to reputation systems, this approach would support individual opinions of users in addition to objective judgement. Collaborative filtering is a more user oriented approach, as each user has a specific similarity to another user, while reputation systems assigns a global trust score to each user.

Since we focus on a platform that manages content, we delimit our focus to be on using collaborative filtering to assess content in a trustless system. Reputation management might be necessary to improve the system overall, but this will not be considered further. Furthermore, reputation management has already been covered considerably with respect to peer-to-peer networks [10] [11] [12].

2.5.2 Trust

A decentralised platform where anyone can join the network, entails a trustless system. Instead of making users responsible for assigning how they trust other users explicitly, we will attempt to exploit that the platform manages reviews of content, and derive a notion of trust based on the rating data. Specifically, we use collaborative filtering to make predictions of how a user would rate given content. This process implicitly derives users with similar rating patterns, which can be viewed as trust, under the assumption that users with similar opinion are more likely to trust each other. Furthermore collaborative filtering can derive a user similarity measure, that can be used to represent trust directly. We delimit the trust aspect to focus on how similarly users rate content.

2.5.3 Communication

We have examined communication protocols, in section 2.4.6, that can be used to facilitate communication and data replication between peers. We deem the communication aspect out of scope of this report, and a specific strategy for providing or implementing communication will not be considered further.

2.5.4 Attack Resilience

The lack of a trusted authority and no explicit trust between users, exposes the system to a range of threads and potential attacks, that must be addressed for the system to remain usable, making attack resilience a core priority. The report will focus on a proof-of-work based approach to mitigate Sybil attacks against a collaborative filtering based system.

2.6 Problem Statement

Many services relies on user submitted content, which makes it necessary to assess its relevance and quality, to provide proper filtering and ranking. Services often accomplishes this by using a rating system targeted at a specific types of content such as movies. The service providers controls the availability of the rating data, and the algorithms used to make predictions.

This project focuses on the feasibility of creating a common platform for managing user submitted reviews of any types of content. The platform should enable applications to integrate a rating system, and provide functionality to assess the relevance of content. We consider a decentralised platform driven by users that should provide unrestricted access to rating data, meaning that the data should be managed by users, and be available for anyone to use.

Based on the analysis and the above delimitation, we define the following problem statement:

> How can a decentralised and trustless rating system, driven by users, be developed to provide a common platform for applications and users to assess content relevance, and enable unrestricted access to rating data?

In addition to this problem statement, we wish to answer the following questions:

- How can the integrity and authenticity of data be ensured using cryptography?
- How can a proof-of-work system and robust collaborative filtering be used to provide resilience to attacks?
- How can collaborative filtering be used to provide rating predictions?

2.6.1 Requirement Specification

In this section we present a list of requirements that has been considered for the system. The requirements have been prioritised, based on their importance, using the MoSCoW method, and as such been divided into the categories: Must have, Should have, Could have, and Won't have.

The system **must**:

- Provide a method to create user accounts
- Provide a method to create metadata that identifies and describes content
- Provide a method to rate arbitrary types of content
- Allow applications to interact with the platform
- Provide methods to assist with content assessment, in terms of retrieving estimates of content relevance to a given user
- Ensure data integrity to prevent false data
- Ensure data authenticity to prevent masquerade attacks
- Be trustless, i.e. not require a trusted authority
- Be driven by users, and allow anyone to operate the platform and join the network
- Be resilient to Sybil attacks, in terms of reducing the feasibility of creating and operating multiple identities
- Be resilient to shilling attacks, in terms of mitigating attempts to manipulate the rating system

The system **should**:

- Provide efficient methods for retrieving data from the platform
- Provide resistance to censorship
- Allow data, in terms of users, metadata and reviews, to be shared between multiple applications

The system **could**:

- Be space efficient
- Prioritise storage and replication of data from similar users
- Represent an implicit trust relationship between peers based on rating similarity

The system **won't**:

- Facilitate a decentralised network
- Replicate data between peers
- Cope with peers continuously leaving and joining the network
- Provide a scalable network, that allows a large number of peers to be connected simultaneously
- Integrate a reputation system
- Mitigate denial-of-service attacks
- Facilitate content search
- Represent and assign explicit trust between users
- Provide content recommendations

3 Design

In this chapter we propose a design of the system, based on the functionality and requirements discussed in the analysis in chapter 2.

3.1 Architecture

We consider a decentralised architecture of the system, where individual peers operates a local instance of the system, and communicates with other peers to replicate data.

We choose to mainly focus on the design of a local instance of the system, operated by an individual peer. The system is separated into multiple components, that provides functionality related to particular requirements of the system. We divide the responsibilities of the components into: system management, data integrity and authenticity, proof-of-work, collaborative filtering, storage management, and communication.

The interactions between the different components is based on a layered architecture. At the top layer a system manager component manages all interaction with the system, and provides the required methods for applications. This component delegates tasks to components of middle layer, which defines all internal logic of the system. At the bottom layer data storage is managed and provides insertion and retrieval required by the other components.

3.2 Components

The system integrates the components, shown in the component diagram on figure 3.1. Each component is responsible for a specific area of functionality in the system, and the component diagram illustrates how the components interact through provided and required interfaces.

The system architecture aims to keep the components loosely coupled, to make it straightforward to substitute them with any desired implementation, and to extend the system with new functionality. As an example, we could use a proof-of-work component based on a different algorithm, or add communication components that transfers data using different types of networks.

3.2.1 System Manager (API)

The *System Manager* component enables applications to interact with the system, and delegates tasks to relevant components. The system exposes a procedure call based application programming interface (API), with methods for managing users, reviews and content metadata, as well as retrieving rating predictions.

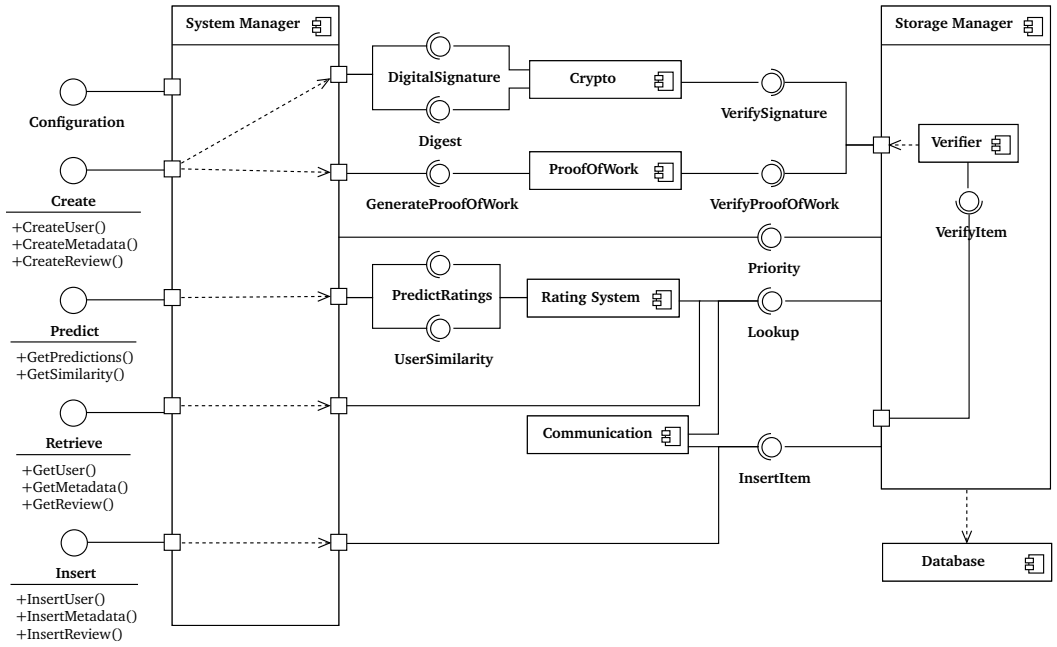

Figure 3.1: Component diagram

The exposed methods can be seen left on the diagram, figure 3.1, illustrated as provided interfaces of the *System Manager* component. The API methods are divided into the different interfaces: *Create*, *Insert*, *Retrieve*, *Predict* and *Configuration*. The dashed arrows inside of the component indicates dependencies. They illustrate the primary interactions involved when calling different API methods, but does not exclude other interactions.

Create

The *Create* interface provides methods for creating objects in the system. When creating a user account, the API delegates operations to the *Crypto* and *ProofOfWork* components. The corresponding components will generate a private and public key pair using digital signature, generate proof-of-work, sign the new user instance and return it.

Metadata for any type of content can be created by providing a description of the associated content.

Reviews of content are created from metadata that describes the content. A review references metadata by a content addressable identifier, which is a digest of the metadata computed by the *Crypto* component. A user assigns a rating to the review, and for the review to become valid, proof-of-work is generated and appended to the review instance together with a signature.

Insert

The *Insert* interface provides methods for inserting objects into the system, such that they can be stored and replicated.

User and review objects are inserted into the system through the *Storage Manager* component, which also verifies that their proof-of-work data and signature are valid. On insertion of metadata, the system checks that there exists a review that references it.

Retrieve

All types of data is stored and can be accessed through the *Storage Manager* component. User information can be queried based on an identifier or public key, and reviews can be queried based on associated users or metadata. Metadata is retrieved based on a review that has a reference to it, or is queried according to associated content.

Predict

Based on the *Rating System* component, the API provides methods for retrieving predictions of content ratings for given user. The *GetPrediction* method is used to retrieve predicted ratings, which can be used to assess the relevance of associated content. It is also possible to retrieve a measure of user similarity, derived form the rating data.

In addition to these methods, it is possible for applications to integrate an external recommendation or filtering system, using the *Retrieval* methods of the API, to compute other metrics or customised predictions from the reviews.

Configuration

The *Configuration* interface provides methods for applications and peers to configure the system. E.g a storage configuration, or communication and network options.

Peers can configure the priority for storage and replication of certain types of content. This can be used to restrict storage to only consider reviews of content, that a given peer is interested in, e.g. reviews supported by a specific application. The configuration can also prioritise a specific user account, based on user similarity derived from the rating system component, which can be used by the *System Manager* component to prioritise content that has been rated by similar users.

Another type of peer, could be an application server that seeks to increase availability of data by replicating all reviews of metadata with a specific type.

The configuration interface is not definite, but leaves room for potentially needed settings, depending on implementation.

3.2.2 Rating System

The *Rating System* component incorporates collaborative filtering, which is used to provide rating predictions, based on an aggregation of reviews in the system. It can also be used to retrieve a measure of user similarity. The predicted rating estimates can be

used to assist applications or users with content assessment. Peers can choose specific subsets of reviews to consider when computing predictions, such as reviews for specific types of content. The time required for computing predictions depends on the amount of rating data considered. Compared to a typical rating system, peers have the possibility to use collaborative filtering that only computes predictions for a single user, and not the entire userbase.

The *Rating System* component depends on the *Storage Manager* component to read rating data when computing predictions.

3.2.3 Storage Manager

The *Storage Manager* components defines how data is stored and retrieved, and provides methods for querying and inserting data. To achieve this, the component utilises a database management system, illustrated by a dependency on a *Database* component, in the component diagram.

The component is primarily used for storing users, reviews and metadata, and is responsible for defining appropriate data indexes to enable efficient lookup of data queried with the API methods.

The *Storage Manager* provides methods to prioritise data. This information is used to decide what data that should be stored, and is also available to the *Communication* component to prioritise acquisition of data. The prioritisation can be based on peer configuration and information from the *Rating System* component, such as user similarity, illustrated in the component diagram.

Verifier

The *Verifier* component is a subcomponent of the *Storage Manager*. It is responsible for ensuring that only valid data is inserted into the system, i.e. user and review instances must contain a valid digital signature and valid proof-of-work data with an acceptable difficulty. For this reason, it depends on the *Crypto* and *ProofOfWork* components.

Metadata is only stored if an existing review is referencing it, meaning that at least one user has reviewed it. This ensures that its relevance can be evaluated, and avoids the possibility of inserting metadata that was created without also generating proof-of-work for a review.

All data stored in the system is subject to verification, such as data from API calls or data received through the *Communication* component. Given this assumption, i.e. that data is verified upon storage, it is possible to retrieve data without validating it, which improves performance and avoids unnecessary storage of invalid items.

3.2.4 Crypto

The *Crypto* component provides a digital signature system and cryptographic hash functions. It has digital signature methods for signing objects, and verifying their signature. It also has methods for generating a private and public key pair, which is used as part of user

account creation. It ensures the authenticity of data in the system, i.e. a claimed user can be verified as the real author of the data. This is essential for objects such as reviews, and serves as a protection against masquerade attacks, and insertion of inauthentic reviews.

The component also provides methods for computing a digest of metadata objects, which is used to reference and identify them, without a risk of data tampering.

3.2.5 Proof-of-Work

The *ProofOfWork* component defines how to compute and verify proof-of-work. It provides methods for performing computational work with a specified difficulty, as well as verifying the validity of proof-of-work data. This involves checking that the proof-of-work data is associated with a corresponding object, such as a review. This guarantees that it is only valid for a specific object, such that proof-of-work data cannot be reused or generated beforehand. Furthermore it is confirmed whether or not the proof-of-work data satisfies an acceptable difficulty. We consider an acceptable difficulty, as a level of difficulty chosen to be high enough to mitigate attacks, but low enough for ordinary users to reliably use the system.

The component makes it time-consuming to produce data, which is considered valid by the system. In particular, it generate proof-of-work data for reviews and user accounts, to increase the time required to rate content and create new user accounts. As described in section 2.4.5, this serves as a mitigation against Sybil attacks.

3.2.6 Communication

The *Communication* component is responsible for facilitating communication between peers. It defines a communication protocol, that considers how peers are connected and controls how and when data is transferred between peers.

The component depends on the *Storage Manager*, which is used to retrieve data and replication priority, and to insert data retrieved by other peers.

As described in section 2.4.6, it could be based on a gossip protocol, however as we delimited the focus on communication in section 2.5, this will not be examined in greater detail.

3.3 Interactions Between Components

To examine the sequence of interactions that takes place between different components. We use sequence diagrams to illustrate some common use cases.

3.3.1 Create User Account

The sequence diagram on figure 3.2, illustrates interactions that occurs between components, when creating a new user account and inserting it into the system. The boxes represents two distinct API calls, meaning that a user is not automatically inserted when creating it. An application that implements the system would typically perform both

actions, since a user account must be inserted to be replicated to other peers. The *GenerateProofOfWork* method is asynchronous, to allow a user to create reviews, before the user is fully created or inserted into the system. Many similar interactions, not shown, occurs when creating and inserting reviews or metadata.

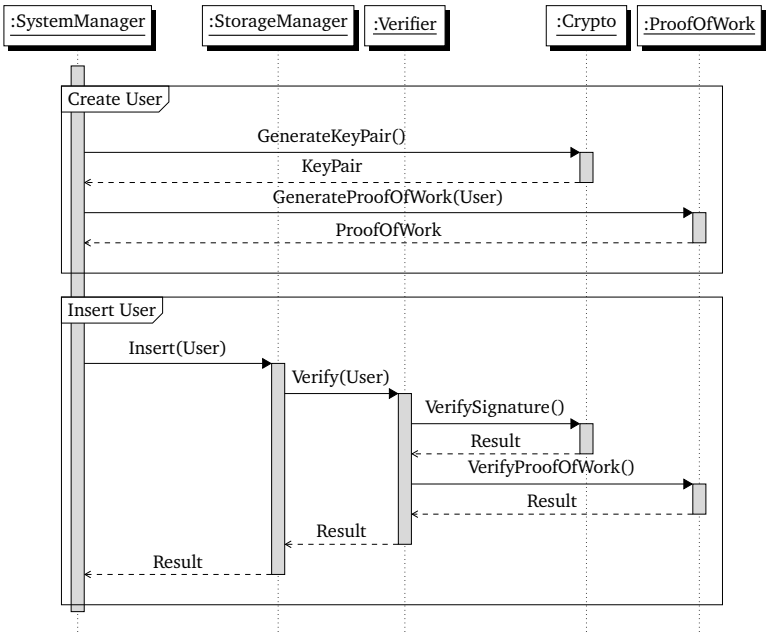

Figure 3.2: Sequence diagram for creating a new user account

3.3.2 Predict Ratings

Figure 3.3 shows a sequence diagram for retrieving rating predictions. First, a lookup of metadata associated with specific content is shown, to illustrate that metadata does not necessarily have to be known beforehand. The *Lookup* method can return multiple instances of metadata, and similarly the *Predict* method can operate on multiple instances of users and metadata. The computation of predictions, illustrated within the small box, could potentially be precomputed and cached, based on the algorithm and prediction model used in the *Rating System* component.

3.4 Classes

This section will describe the fundamental data types or classes of the system. The system is generally defined based on the classes: *User*, *Review* and *Metadata*. A *Peer* class is used to represent peer configuration. A diagram of the classes and their relations are shown in figure 3.4.

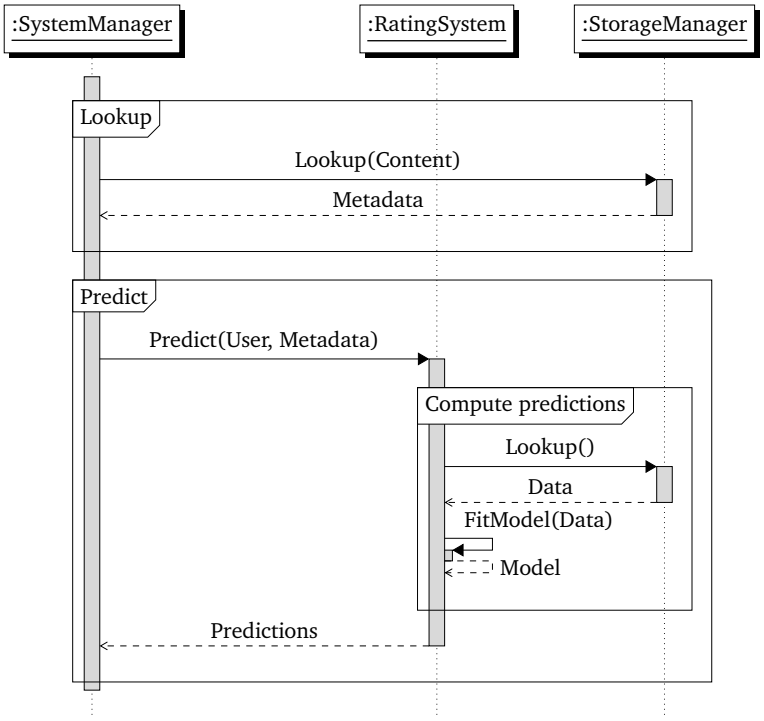

Figure 3.3: Sequence diagram for retrieving rating predictions

3.4.1 Metadata

The *Metadata* class describes specific content and how to identify it.

The *contentID* attribute holds a reference that can be used to identify the actual content, such as a file or book. The *identifierType* attribute indicates how to interpret the *contentID* attribute. E.g. a file could be identified by a digest of its data, in which case the *identifierType* attribute would describe the hash function used to produce the digest. In case the content is a book, the *identifierType* could be ISBN[1], and the *contentID* would contain the ISBN number of the book. An application is responsible for choosing type values and how to use them, but we recommend to use unique and persistent identifiers that does not change over time, such as content addressable identifiers, described in section 2.4.1. References such as a URL are therefore undesirable, as the source content can change.

Ideally, applications will agree on using the same identifier types, to be able to share data. To assist with this, we recommend using standardised identification schemes. Examples could be the uniform resource name (URN) scheme [22], that can be used to represent identifiers within defined namespaces, or the digital object identifier (DOI) system [23], that can provide persistent identification of objects for use on digital networks.

Applications are responsible for locating the actual content based on the *contentID*. This could for example be done, by performing a lookup in a related database, or use a

[1]ISBN: International Standard Book Number

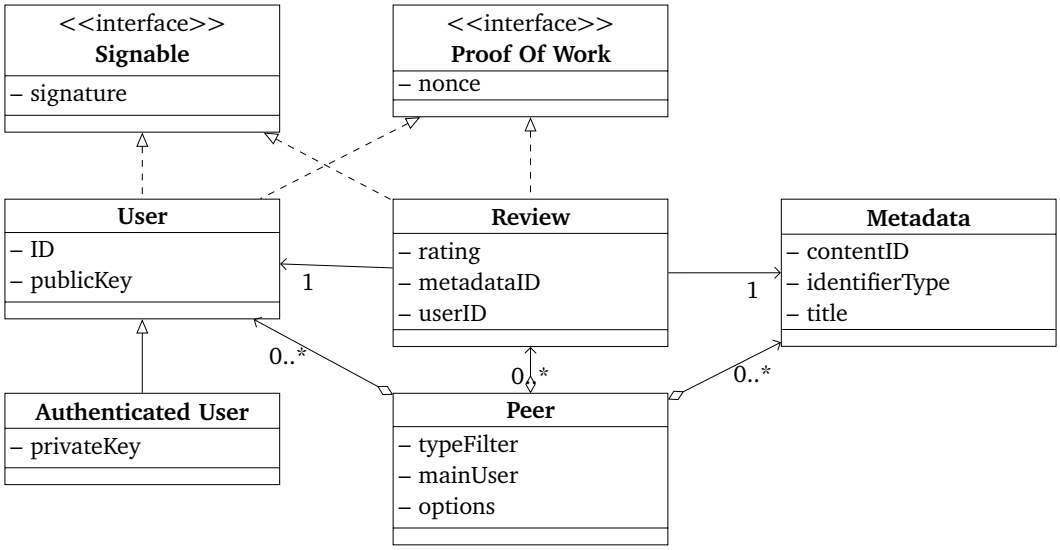

Figure 3.4: Class diagram of the classes in system

content addressable system such as BitTorrent to acquire it.

As shown in the class diagram, a metadata class also has a *title* attribute. This is used to describe the content, and allows users to review that this information is correct. Many additional attributes could be included to represent content, such as the type of data, time of creation, and keywords. These are not included in the diagram, because not all attributes applies to all types of content, but more importantly because more attributes makes it more likely that users or applications will create nearly identical metadata for the same content, but with minor differences for the set of attributes. A single *title* attributes is sufficient for most cases, and can contain additional information if desired. This is not an ideal approach, but we will not consider appropriate management of content attributes further in this report. Different metadata can refer to the same content, but incorrect metadata should encourage users to provide a low rating, compared to content with correct metadata.

The *identifierType* attribute can also be used to decide what type of metadata and associated reviews that are retrieved and stored.

3.4.2 Review

The *Review* class represents a given user's assessment of content and associated metadata, in form of a rating.

A review has a reference to a specific user and a specific instance of metadata, as indicated by the association relationship in the class diagram on figure 3.4. In order for a review to be secure against data tampering and masquerade attacks, it is signed using digital signature, described in section 2.4.2. Furthermore, it includes proof-of-work, described in section 2.4.5. This is represented by the *Signable* and *ProofOfWork* interfaces, in the class diagram. The interfaces can be implemented by a class, to add support for

assigning a digital signature and proof-of-work to instances, however it is not important to implement it strictly this way.

A valid review includes a rating, a content addressable identifier to related metadata, a user identifier, proof-of-work, and a signature. The review is signed after computing and adding the proof-of-work data. If this was not the case, a malicious peer could substitute the proof-of-work data, by recomputing it with a different difficulty.

3.4.3 User

The *User* class represents a user account, and contains a unique identifier and a public key. The *publicKey* attribute is used to verify the authenticity of reviews. An authenticated user, represented by the derived class, contains an additional *privateKey* attribute, used for signing objects, such as reviews.

Similar to the *Review* class, the *User* class implements digital signature and proof-of-work interfaces. Proof-of-work is added to a user account to mitigate attempts by a single person to create many user accounts, as described in section 2.3.3.

A simplification of the class could be to use the public key as identifier, which would avoid the need to lookup the public key when verifying reviews. However, in case an implementation uses large public keys, a separate user identifier can be used to reduce the size of reviews.

Additional user information could be represented by this class as well. To update the user information, proof-of-work with a larger difficulty could be generated, and used to determine the most recent set of information.

3.4.4 Peer

The *Peer* class represents the current configuration and state of a peer. As described in section 3.2.1, a peer can be configured with information about what types of content to consider for replication and storage, indicated by the *typeFilter* attribute. The *mainUser* attribute indicates an optionally prioritised user account. The class also holds information about other desired options in the system, such as maximum space used for storage, and connection information.

3.5 Data Structure

When data in the system is replicated between peers, a data structure for organising the data can be used to improve the efficiency of transmitting data, in terms of reconciling differences in data sets between peers. Furthermore, it can provide methods for prioritising or selecting specific content to be replicated.

A simple approach could be to make each user assign an ordering to each their reviews. When two peers are communicating they can then determine which peer has the newest review, based on the ordering, and then transfer the missing reviews to the corresponding peer. However, this approach is not very flexible and requires that all reviews are signed

and verified as they arrive, which could be inappropriate for peers that needs to acquire a large amount of data.

A more advanced approach could be to utilise a consensus based blockchain technology, to store a chain of reviews. However, because this is associated with a significant overhead, and because global and persistent storage of data is not a requirement for this system, we will not consider this approach further.

3.5.1 Simple Blockchain

A simple blockchain data structure could be utilised to organise a list of reviews for each user, with each item in the list referencing the previous item by its cryptographic hash. This structure forms an append-only data structure, that could be reconciled between peers, by comparing their most recent entries. A peer would only have to verify the most recent review, since that would guarantee that the author has endorsed the entire chain of reviews linked to it.

A disadvantage of this approach, is that a user could fork the chain, such that different reviews reference the same parent review. The system would need to define rules for how to handle the detection of inconsistent chains of reviews, and how to determine which version of the chain to consider correct. This would however make it possible for a malicious user to remove previous reviews, by creating a new chain. Because the detection of two inconsistent chains, signed by the same user, guarantees that the user must have forked the chain, it could be decided to discard the chains and invalidate the user entirely.

However, this means that if a user loses the list of reviews, and has to retrieve it again from the network, getting an older version of the list, could cause the forking problem, since there is no way to detect if more recent reviews exists.

3.5.2 Merkle Tree

Another approach is to utilise a Merkle tree. A Merkle tree is a tree in which each node contains the cryptographic hash of its child nodes. The leaf nodes contains the hash of a data block, which in the case of this system could be a review object.

Similar to a blockchain, the tree structure makes it possible for peers to detect new entries. In addition, any changes in the tree can be determined by traversing it and comparing hash labels, which allows peers to retrieve reviews in any order. Furthermore, it is also possible to determine if a given review is part of the tree, by computing the hash of reviews up through the tree. To verify the entire tree, only the root node would have to be signed.

Similar to a blockchain, inconsistent trees, in which none of them are a subset of the other, can be created and signed by a malicious user, requiring rules for how to handle this.

3.5.3 Bloom Filter

An alternative to using a hash chained data structure, is to replicate the objects individually, as mentioned in the beginning of this section.

The disadvantage of requiring each peer to verify every review, might not be a major issue if verification can be done efficiently. However, replicating metadata objects individually, when they are needed by a peer, cannot easily be done by ordering them, since peers could have different subsets of them.

A possible solution is to use a bloom filter, which can encode which objects a peer has, with some probability of false positives. E.g. a peer could send a bloom filter, which is a compact and approximate representation of which items are in the metadata list, to another peer, which could check which metadata each of the peers are missing. The problem of false positives, means that some metadata would not be detected as missing, and thus replicated, however by computing different bloom filters for each transaction, all metadata should be synchronised over time.

A benefit of this approach is that reviews cannot easily be removed by malicious users, since they will be replicated from peers that have already acquired them. This avoids the need to detect possible inconsistencies, such as forks of a review chain, and makes it possible for users to insert new reviews without the possibility of conflicts, even if the user does not have access to the most recent list of reviews.

3.5.4 Comparison

Blockchain and Merkle tree data structures allows efficient reconciliation of data, and requires that only a few signatures is verified for each transaction. However, since verifying signatures is not a major issue, we do not consider these data structures as the most suitable choices.

Signing each object, and defining and ordering of them, could be sufficient to enable peers to efficiently replicate data in the system. A thorough evaluation of the efficiency required for replication, is needed to decide the most suitable approach. As we regard implementation of the communication aspect out of scope, as mentioned in section 2.5, we do not make a definite decision. However, because the bloom filter approach provides a more flexibility and has some interesting characteristics, with regards to this system, we currently consider this approach as the best candidate.

4 Implementation

In this section we examine possible strategies and existing methods that can be used to implement the system components.

4.1 Crypto

The *Crypto* component implements cryptographic functionality. In particular, cryptographic hash functions and a digital signature system.

4.1.1 Cryptographic Hash Function

As described in section 2.4.1, a cryptographic hash function can be used to produce a digest or hash of data. This can be used to ensure data integrity when transferring data between peers. In this system, a hash function is used to produce a content addressable identifier for metadata, that can be referenced in reviews, such that peers can verify that any given instance of metadata is certainly what is being referred to.

The National Institute of Standards and Technology (NIST), has published a family of cryptographic hash functions called secure hash algorithms (SHA). At the time of writing, the SHA-2 [24] and SHA-3 [25] families are considered secure, and has no published examples of collisions. SHA-3 is the most recent member of the SHA family, and was developed and selected as part of a hash function competition[1], held by NIST. Another hash function, BLAKE2 [26], based on one of the five final candidates in the competition, is claimed to be faster and at least as secure as SHA-3.

It is crucial that the hash function has a high level of security, and fulfils the properties mentioned in section 2.4.1. Depending on the implementation, it might be beneficial to utilise a fast hashing function, such that data can be hashed and verified often with low processing overhead.

Because of its advantageous speed and security properties, we have selected BLAKE2 for the proposed implementation.

4.1.2 Digital Signature

Digital signature is used to ensure the authenticity of user related data, primarily considering reviews. To reduce the amount of data, that peers need to store and transmit, it would be preferable for the system to use signatures with a relatively low size. This is especially a concern, when considering an approach not based on chaining, given that every review must contain a signature.

[1] `https://csrc.nist.gov/projects/hash-functions/sha-3-project`

The ECRYPT benchmarking of asymmetric systems (eBATS) project [27], has measured the performance of several public key systems. We select some of the systems, shown in table 4.1, with a focus on the size of signatures and keys in bytes, and the speed of verifying a signature in terms of cycles.

System	Type	Signature	Secret key	Public key	Verify (cycles)
mqqsig160	MQQ	20	401	206112	25014
donald512	DSA	40	84	64	127050
ronald512	RSA	43	512	64	22528
ed25519	EdDSA	64	64	32	150194

Table 4.1: Excerpt of eBATS benchmark results [27].

The MQQ-SIG [28] system, based on multivariate quadratic quasigroups, has the smallest signatures, requiring 20 bytes for the 160-bit variant. It is among the fastest for both signing and verifying. The public key is large which makes it preferable to use a separate and smaller user identifier to reference reviews.

The donald system is based on the digital signature algorithm (DSA), and uses a relatively small signature and public key for the 512-bit variant.

The ed25519 [29] signature system, based on elliptic curves, has the smallest public key. Its verification speed is comparable to donald512 for single signatures, but can provide faster verification in batches.

The RSA-based ronald system has the fastest verification. It has the largest secret key, however this is not an issue since it is never transferred and only stored by the user it represents.

All of the considered systems, are fast and produces relatively small signatures, making them suitable for adaptation and implementation of the *Crypto* component. Despite having the slowest verification speed and largest signature, the ed25519 system is still promising because it has a higher level of security[29] than the other examined systems. It also uses a 32 bytes public key, which is reasonable to use directly as a user identifier, as discussed in section 3.4.

We also consider the MQQ- and RSA-based systems to be suitable candidates, given their high speed and small signatures. They exists in variants with higher levels of security, usually at the cost of speed and an increased size of the public key. This can be dealt with by using smaller user identifiers to reference reviews, e.g. based on a digest of the public key.

For the proposed implementation, we have selected and used the ed25519 system.

4.2 Proof-of-Work System

As described in section 2.4.5, a proof-of-work system can be used to make Sybil attacks more difficult to execute. In particular, memory-bound proof-of-work systems can be especially cost-ineffective to an attacker.

Key derivation functions such as scrypt [30] and Argon2 [31] are examples of memory-bound algorithms, focusing on a time-memory trade-off, that has been used in proof-of-work systems. Examples of memory-bound proof-of-work systems that has been proposed or used to implement cryptocurrency systems are:

- Cuckoo Cycle [32]: based on finding cycles in random graphs.
- Equihash [33]: based on the generalised birthday problem.
- MTP-Argon2 [17]: based on Merkle tree proofs.

All of the listed systems can potentially be used to implement the proof-of-work component. In particular, the MTP-based system is promising, because it can be configured to require large amounts of memory to generate the proof-of-work data, while also performing verification fast and efficiently. This appears to fulfil the criteria required by the system, however a thorough evaluation of the different systems is out of the scope of this report, and a specific system will not be considered for implementation. In the proposed implementation, we use compute-bound proof-of-work, based on hashcash, which illustrates the purpose of the component, in terms of time taken to perform the computation.

We have considered various approaches to make the implementation of proof-of-work cause minor interference to ordinary users, while remaining sufficiently disadvantageous to attackers. Performing the required computation as a background task with low priority, enables the user to use other applications without experiencing significant slowdown. In case the user needs to use a memory intensive application, the proof-of-work computation might have to be stopped completely, depending on the required amount of memory. However, only a fraction of the available memory in modern computers should be necessary to provide suitable attack resilience.

A peer must be active for longer periods of time, to ensure that computation of proof-of-work does not queue up over time. We assume this is not a problem for most users.

A user can continue to use the system while proof-of-work data is computed. This includes computing rating predictions, based on pending reviews, as replication of the review to other peers, is the only operation that must be delayed until the proof-of-work computation is complete.

4.3 Storage

The storage manager component is responsible for handling requests for insertion and retrieval of data. The component defines how data is stored in order to optimise aspects such as performance and space usage. This is achieved by delegating requests to an internal database management system. The database is implemented as an internal part of the system, only needs to interact with the storage manager component. For this reason, the performance of the database is more important than the exact type. Different peers could technically be using different databases internally, as the storage manager component is responsible for providing a compatible interface.

Given the explicitly defined structure of the few essential objects in the system, described in section 3.4, we consider a typical relational database that uses a structured query language (SQL). Specifically, we have based the implementation on the self-contained database engine SQLite[2].

4.3.1 Database Schema and Indices

The database schema uses tables based on the classes, described in section 3.4.

The performance of most database management systems can be improved by defining indices for data that is queried often.

User instances is indexed by the unique user identifier attribute. This attribute, which is based on or equivalent to a public key, should not to be confused with an internal identifier, used for performance reasons to reference users from different table entries, such as a review.

For metadata, indices is defined for the content addressable metadata identifier, in our case a BLAKE2 Hash, and the content identifier. This will allow looking up metadata referenced in reviews, as well as retrieving existing metadata for given content.

Reviews have indices for the user identifier and the metadata identifier, to allow looking up all reviews associated with given metadata, or indirectly given content, as well as retrieving reviews created by given users.

[2]https://www.sqlite.org/

5 Evaluation

In this section we introduce experiments that have been conducted to evaluate different aspects of the system, in terms of scalability and attack resilience. The scalability aspect examines how space requirements expands as the size of the system grows. Attack resilience is evaluated based on experiments that examines the impact of proof-of-work in relation to the effect of an attack, using different types of collaborative filtering.

5.1 Scalability of Disk Space Usage

This section will evaluate the required space needed by each peer in relation to the size of all data in the system. The required storage space needed for a peer to operate the system, depends on the amount of data that exists in the network, and the peer configuration, mentioned in section 3.2.1.

As the amount of data in the system grows, concerning the number of users, reviews and metadata objects, peers generally need to store more data. They can however be selective about what they store, or how much they store, in order to reduce use of disk space. E.g. a peer that prioritises a specific user account, as mentioned in section 3.2.1, can be configured to primarily store reviews from similar users, and a smaller amount of reviews from randomly selected users.

To simulate realistic space usage in the system, SQLite database engine has been utilised to store the data, and been configured with the indexes mentioned in section 4.3. We will consider the space required to fully replicate all data in the system, and the space required for a minimal representation that only keeps identifiers and relations. A configuration that prioritises certain reviews will require an amount of space between these values.

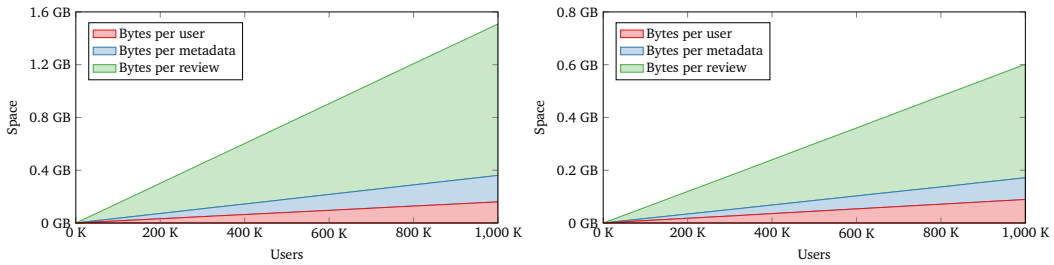

Figure 5.1: Usage of space in relation to the number of users

Figure 5.1 shows two stacked area charts, that illustrates the space usage of the database in relation to the number of users. The scenario considers a case where there is always exactly one metadata object per user, and 10 reviews per user. It can be seen that usage of space grows linearly in respect to the number stored items.

The left chart shows the space required when storing all data in the system, and the right chart show the space required when storing a minimal representation of the data, keeping only the public key of users, the content addressable identifier of metadata, and the rating of reviews. After the data has been verified, this is sufficient for the system to function, however none of the data can be replicated to other peers.

The minimal representation is approximately half the size of storing the complete data set, as seen in table 5.1.

Item	Minimal	Full
User	88.68	159.8
Metadata	82.53	200.61
Review	43.05	114.78

Table 5.1: Average space usage per item in bytes

Both configurations, distributes space similarly between users, metadata and reviews. User and metadata instances uses a comparable amount of space, while a review instance takes up relatively little space, recall that the figures illustrates 10 reviews per user.

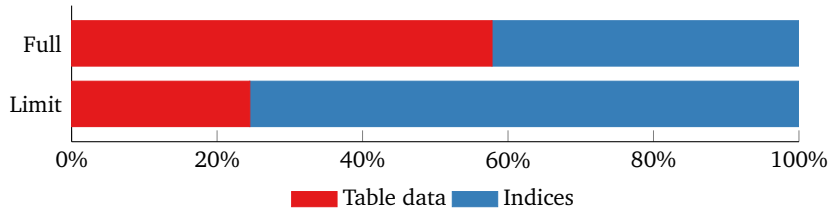

Figure 5.2: Distribution of space between table data and indices

Figure 5.2 shows the partitioning of data used for tables and indices. A significant amount of the stored data is used for indices. When storing a minimal amount of data, by removing proof-of-work data and signatures, the indices accounts for more than half of the stored data.

In case the data size in system becomes very large, other strategies can be employed. E.g. indices can be removed if lookup time is not a high priority, or users that has not been active in a given interval can be removed.

Overall, this scenario has relatively lenient space requirements, and the described methods of reducing space usage should be sufficient to support most use cases. Considering that a peer, in this scenario, can support up to a billion users with 1.6 TB of storage space, storing the complete amount of data, it is unlikely that space usage will become an issue.

5.2 Attack Resilience

To determine acceptable proof-of-work difficulties, experiments have been made to examine how the rating predictions of the system can be affected by an attack in a certain scenario.

Specifically the accuracy of user-based collaborative filtering algorithms are evaluated, in terms of the number of reviews and user accounts added by an attacker to affect the predicted rating of a specific item, compared to the number of affected users, that obtains a positive rating for the item. The algorithms examined are common versions of collaborative filtering based on k-nearest neighbors (k-NN) and singular-value decomposition (SVD).

5.2.1 Scenario

We consider a system consisting of 1000 instances of metadata, and 1000 user accounts, where each user have rated approximately 30 items. All ratings are either 1 or 0, indicating that an item has been verified as being correct, or marked as false, respectively. All users are genuine and consistent, in the sense that they always rate items correctly, which means that no item has received conflicting ratings, i.e. all ratings that an item has received has same value. To simulate different interests among the users, they are divided into 10 distinct groups, such that 80% of the reviews are assigned to items chosen at random, within a subset of metadata assigned to the corresponding group of users, and the remaining 20% of reviews are assigned to items chosen at random in the complete set of metadata.

We consider a scenario where an attacker has added a new false item, i.e. metadata that incorrectly describes the associated content, e.g. a virus described as a another popular file.

The attacker creates multiple user accounts, called attack profiles, that all creates a review which rates the false item as being correct, i.e. with a rating value of 1. Furthermore, the attack profiles creates reviews for a fixed number of existing items chosen at random, known as filler items, and rates them correctly, to increase similarity with genuine users.

This scenario should make it difficult to obtain a satisfactory prediction for the false item, because it only has received positive reviews, and attack profiles and genuine users agrees on reviews of all other items.

The effectiveness of the attack is evaluated by considering the percentage of genuine users that obtains a positive prediction for the false item, with a rating value above 0.85, i.e. within a margin of 0.15 from the maximum value of 1. This represents a prediction that a user should ideally have high confidence in.

The collaborative filtering algorithms are fitted on 90% of the reviews. The remaining 10% are used as evaluation data to measure the accuracy of the prediction model.

5.2.2 K-Nearest Neighbours

The k-nearest neighbours (k-NN) algorithm is a memory-based collaborative filtering algorithm. The algorithm is configured to use cosine similarity with a minimum support of 3, meaning that users must have created reviews for at least 3 common items, in order to be considered similar. This is done to avoid high similarity between users that only has a few items in common. However, because users always agree, they will be considered 100% similar if they have at least 3 reviews in common.

With this setup, approximately 90% of the predicted ratings of the evaluation data, are predicted correctly within a margin of 0.15. Within a margin of 0.50, i.e. where predictions are rounded to nearest whole number (0 or 1), the evaluation data perfectly predicts the evaluation data.

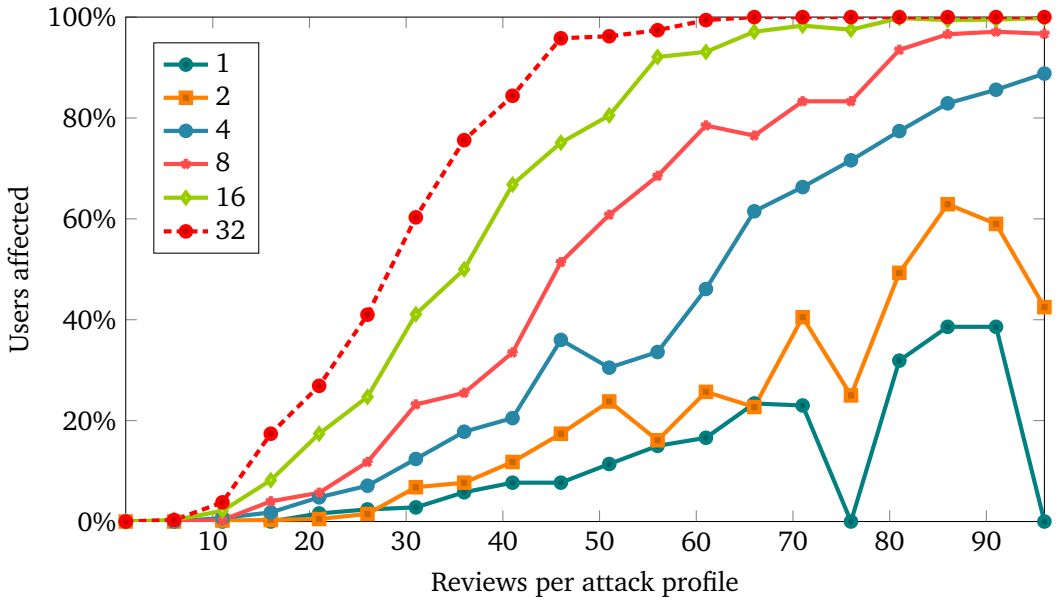

Figure 5.3: Effect of attack when using k-NN based collaborative filtering

Figure 5.3 illustrates six experiments, with each series representing an experiment with a given number of attack profiles inserted into the system. The number of affected users is displayed on the y-axis, and the number of reviews added by each attack profile on the x-axis.

This means that for each point on the x-axis, an experiment adds a number of reviews in proportion to its number of attack profiles, e.g. the blue series with 2 attack profiles, adds twice as many reviews as the red series with 1 attack profile.

It can be seen that it is generally more effective to add reviews, than to add attack profiles to affect the same number of users. It is disadvantageous, that an attacker reliably can increase the effect of an attack, with a low number of attack profiles. However, if attackers are motivated to create more reviews and less attack profiles, it can be easier to filter out the attack reviews, because they are associated with a relatively few user

accounts. This would require more sophisticated methods, such as shilling detection mentioned in section 2.4.4.

5.2.3 Singular-Value Decomposition

A commonly used model-based approach to collaborative filtering is singular-value decomposition (SVD). In this experiment, we configure the SVD algorithm to use 20 factors, and optimise it using stochastic gradient decent for 20 epochs.

On the left plot, of figure 5.4, it can be seen that when using the SVD based approach, the effect of the attack seems highly related to the number of attack profiles, and seemingly unaffected by the number of reviews per attack profile. Even when the attack profiles have made very few reviews, i.e. have low similarity with genuine users, the genuine users are still affected, given enough attack profiles. This is probably the effect of dimensionality reduction, which is used for the SVD approach, compared to the k-NN approach that was based on a similarity measure.

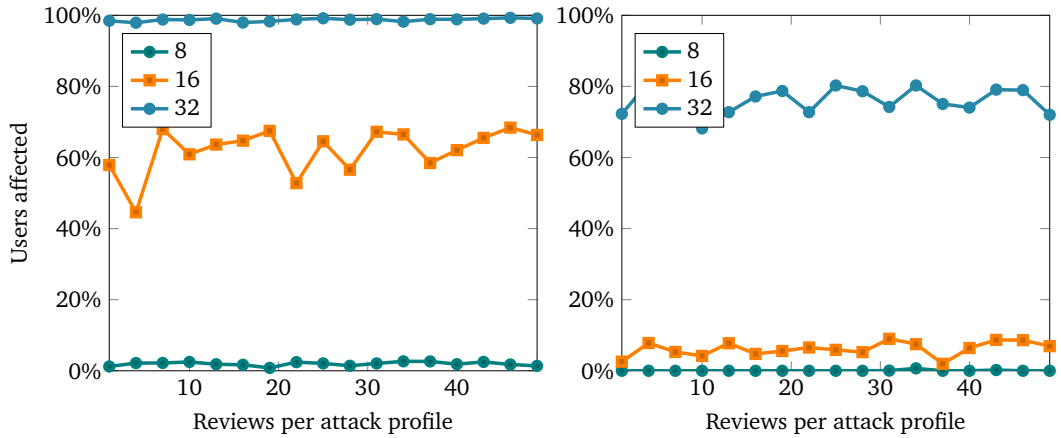

Figure 5.4: Effect of attack when using SVD based collaborative filtering

The plot on the right of figure 5.4, illustrates a similar setup, which includes negative reviews of the false item, added by 3 genuine users chosen at random. This represents the effect of the SVD based approach, when not only attackers have rated the false item. The number of affected users is reduced significantly, for the experiment with 16 attack profiles. The effect of using 32 attack profiles is also reduced, but still affects up to 80% of the users.

5.2.4 Proof-of-Work Difficulty

To compare how much time it would require to generate proof-of-work for the user accounts and reviews created for an attack, we consider an example case, illustrated on figure 5.5, that requires certain proof-of-work difficulties. Each review is assigned proof-of-work with a difficulty that on average would require 12 minutes to generate.

User accounts is assigned proof-of-work that on average would take 8 hours to generate. We choose these values, to consider a situation that requires relatively much work for genuine users, but remains practicable for using the system. We mainly focus on the growth in time required to perform the different attacks, and not the exact values.

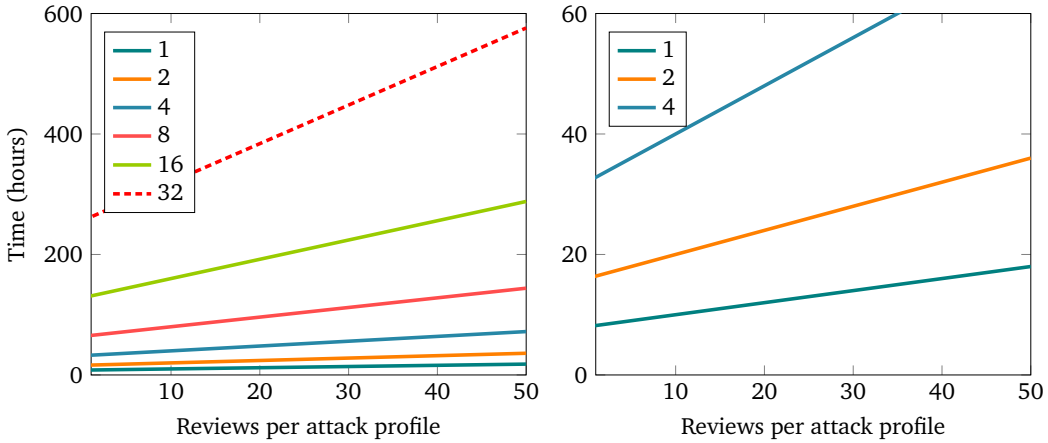

Figure 5.5: Time for generating proof-of-work for attack profile reviews

The y-axis illustrates the total amount of time, it would take to generate proof-of-work for a number of reviews shown on the x-axis. The different series begins from the initial time needed for creating the corresponding number of attack profiles. The right plot more clearly shows time needed for experiments with fewer attack profiles.

Each series grows proportionally to the number of attack profiles, given that more reviews, and thereby more proof-of-work, are created for each profile. In this scenario, attack profiles are more expensive to create than reviews, not only because the proof-of-work difficulty is higher for creating user accounts, but also because a number of reviews is created for each attack profile. However, if the effect of the attack can be increased more quickly by adding attack profiles rather than reviews, which appears to be the case for the SVD based method, that approach is more effective.

When using the k-NN based approach, seen in figure 5.3, an attacker could affect 40% of the users, with a single attack profile by creating 90 reviews. In this example case that would take around 26 hours. This is an effective attack for a relatively short amount of time. In comparison the same effect could be achieved with 2 attack profiles each creating around 70 reviews. This would however take significantly more time, of around 44 hours.

5.2.5 Evaluation of Results

In this section we evaluate the results and consider how we can use them.

Validity of Experiments

Because the conducted experiments considers a very limited set of cases, the results might not generalise to a realistic setting. As described in section 5.2.1, only a single scenario is considered. In the scenario each item has been rated by 30 users on average, which does not consider the more realistic case, in which most items has received few reviews. The chosen scenario makes it straightforward for attackers to increase their similarity with genuine users by rating randomly chosen items.

We have considered the effect of an attack in relation to all users, it could be relevant to make experiments that consider the effect on individual groups.

The selected scenario considers 10 different groups of users, that approximately rates the same number of items, with each group primarily making reviews for a corresponding subset of items. In a more realistic scenario, it is common that a certain subset of items are more popular, i.e. receives more reviews that other items. In that case, attack profiles that rates the most popular items, can achieve high similarity with a large number of users.

Collaborative Filtering and Proof-Of-Work

In the scenario covered by the experiments, the false item is exclusively rated by the attacker. This causes unsatisfactory predictions, when adding enough attack profiles and reviews, and a number of genuine users must provide reviews that rates the false item negatively, to improve rating prediction. We consider both the k-NN and SVD based approaches insufficient to avoid potential attacks, based on the necessary proof-of-work difficulty and effectiveness of attacks.

Robust Collaborative Filtering

In order to sustain an acceptable proof-of-work difficulty for regular users, the collaborative filtering approach should take attack resistance into account, and different methods for detecting and mitigating shilling attacks should be evaluated. In section 2.4.4 we introduced robust collaborative filtering, and examined the VarSelect algorithm, which is a viable candidate for detecting shilling attacks.

Experimental results of the performance of a VarSelect based algorithm, called VarSelectSVD, is presented in [13]. The experiments is based on a dataset with 6040 users and 3942 movie items. The algorithm generally performs better than a common SVD based algorithm. The performance of the algorithms decreases as the size of the attack is increased.

We consider experimental results from performing an average attack, as mentioned in section 2.3.3, The experiment uses an attack size of 1%, corresponding to around 60 attack profiles, and a filler size of 7%, corresponding to around 276 reviews. The time needed for generating proof-of-work for this case, is 480 hours for creating the attack profiles, and 3312 hours for creating all the reviews, which we consider sufficient to discourage most attackers. The hit ratio is the percentage of users that are affected by the attack,

meaning that an attacked item will appear in a user's list of recommendations. Compared to an ordinary SVD algorithm the hit ratio goes from 10.67% to 0.14% with VarSelectSVD. If the hit ratio compares to the percentage of affected users in our experiments, this is a promising improvement. However, since this has not been evaluated, we also consider the prediction shift, which is the average change in the predicted rating of an attacked item, for all users. Compared to an ordinary SVD algorithm the prediction shift goes from 1.24 to 0.42 with VarSelectSVD. The means that the prediction for the attacked item is impacted around 3 times less when using VarSelectSVD. An attack with less attack profiles may still be feasible in terms of the required proof-of-work, but when considering that the performance of the algorithm does not decrease for smaller sizes of attacks, the improved prediction shift could be sufficient to discourage most attacks. Experiments using an implementation of the algorithm is needed to evaluate the performance of the algorithm for our purposes.

It should be noted that if an application is mainly focused on providing recommendations, we consider the performance of this algorithm to be sufficient, based on the improvement of the hit ratio. However, this system also focuses on reducing the predicted rating of an attacked item to a degree that would avoid that a user obtains an inappropriate estimate when assessing the item, even if the predicted rating is not high enough to be recommended. Specifically, as mentioned in the experiments, the prediction should not become within a margin of 0.15 from the incorrect rating, meaning the false rating provided by attack profiles.

Reusing Attack Profiles and Reviews

The system discourages attacks through the use of proof-of-work, however there is currently nothing that mitigates reuse of the attack profiles. Strategies for users to filter out attack profiles could be based on explicit trust, a reputation system, or an extension to the rating system. By utilising such an approach, the effect of an attack that reuses attack profiles can be reduced significantly. It could be difficult, and potentially impossible for an attacker to determine the effect of a subsequent attack, given that it is unknown which and how other users trusts the attack profiles. This makes it more reliable for an attacker to create new attack profiles and reviews, which can be assumed to not be blocked or ignored by other users, however this approach is also much more expensive. We consider a strategy to avoid reuse of attack profiles as an important consideration for the system, but an exact implementation and evaluation of this is out of the scope of this report.

6 Conclusion

This report has examined how to create a common rating platform that can be used by applications to allow users to submit reviews of arbitrary types of content, and utilise rating data to assess the relevance of content for given users. The solution considers a decentralised and trustless system that allows users to share rating data and provide unrestricted access to it.

In section 2.6 we defined the following problem statement:

> How can a decentralised and trustless rating system, driven by users, be developed to provide a common platform for applications and users to assess content relevance, and enable unrestricted access to rating data?

In the following paragraphs we break down this statement, and answers how we have approached the individual requirements.

To provide a common platform that allows applications and users to interact with it, we defined a procedure based API in section 3.2.1. The system is based on three essential data types: user, metadata and review. The API provides methods to create and retrieve all types of data in the platform. Data inserted into the system is shared and can be used by any application.

As described in section 3.4, the metadata type can describe any type of content, by providing abstractions for how to identify and interpret the described content.

To achieve a trustless system, independent of an authority, we use cryptography to ensure integrity and authenticity of data in the system. In section 4.1.2, we examined suitable digital signature systems, that can be used to represent user identities and sign reviews. Users can individually verify that any data originates from a given user, which allows peers to relay data between each other, without a risk of undetected insertion of inauthentic data. We use a cryptographic hash function to assign content addressable identifiers to metadata, which ensures data integrity and makes it unfeasible to forge an association between a review and metadata. We considered different cryptographic hash functions in section 4.1.1, and found BLAKE2 to be suitable for this purpose.

The requirement for the system to be user-driven, is enabled by a decentralised architecture and trustless design. The system design, described in chapter 3, considers a decentralised architecture, such that the system is operated by multiple peers. Because the system is trustless, anyone can join the network and become a peer to replicate and insert reviews. We have not evaluated the communication and network aspect of this design, however, in section 2.4.6, we examined gossip membership protocols that can be used to implement it.

The system provides unrestricted access to rating data, in the sense that the data is public and can be accessed by connecting to an active peer. The availability of data depends on the number of active peers in the system, i.e. availability is improved when more peers are available to communicate and store the data.

To assist applications with content assessment, the system integrates a rating system component, that uses user-based collaborative filtering to predict how users would rate content. This allows applications to estimate the relevance of content to a given user. As described in section 3.2, collaborative filtering is also used to derive a measure of user similarity, which can be used by peers to prioritise data from similar users, in terms of storage usage and replication.

Resilience to attacks is an important requirement, because trustless systems are susceptible to Sybil attacks, which we examined in section 2.3.3. Proof-of-work is a core part of the platform and serves as the primary approach to mitigate Sybil attacks. The system requires that proof-of-work is computed and assigned to all reviews and user accounts, with the purpose of making attacks more difficult and less profitable for an attacker to perform. We have examined memory-bound proof-of-work systems, in section 2.4.5, which has properties that make them suitable candidates to discourage attacks. In section 4.2, we argue that the MTP-Argon2 proof-of-work algorithm fulfils the requirements of the system, however we have not implemented and evaluated the algorithm, which we deem necessary to examine its applicability for this system. In section 5.2, we have evaluated collaborative filtering techniques, based on k-NN and SVD, for a given scenario and deem proof-of-work alone insufficient, because rating predictions can be manipulated substantially with a relatively few number of user accounts and reviews. We have examined results of a robust collaborative filtering algorithm by [13], discussed in section 5.2.5, and by comparing how it to our experiments, we deem proof-of-work in combination with robust collaborative filtering, able to avoid a large range of potential attacks. The exact effectiveness of such combination depends on the algorithms used, and requires further evaluation.

Scalability, in terms of space usage, has been examined in section 5.1, and different approaches have been proposed to reduce space requirements. However, because of the relatively low amount of space needed per data object, and a linear growth with respect to objects in the system, space usage per peer is not a significant issue, and typical peers would be able to store the complete set of data under expected scenarios. The system design allows peers to primarily retrieve and replicate types of data that they are interested in, or is needed by applications they use. Scalability, in terms of network size and communication of data, has not been considered in this report, but we find the protocols mentioned in section 2.4.6, to be promising candidates for implementing a scalable solution. Compared to rating systems commonly used by service providers, this system makes peers responsible of performing collaborative filtering. The computation time for computing predictions can be decreased by selecting a smaller subset of reviews, and the possibility of only computing predictions for a single user.

Based on an examination of proposed methods and existing technologies, we argue that a trustless and decentralised rating platform is feasible to implement and operate. An implementation of such system is not straightforward an requires a combination of different technologies to facilitate a robust and functional system. Similar to other rating systems, a relatively large number of active users is needed for it to be worthwhile, in order to have enough rating data available to provide proper rating predictions. We deem Sybil attacks to be the primary thread to the system, and have examined a proof-of-work

based mitigation approach in this report. We have focused on providing a foundation of the platform by examining and defining essential components. Future work includes an evaluation of a complete implementation that integrates all of the required components, and an evaluation of usage in a realistic setting, to determine the effectiveness of a possible solution.

6.1 Future Work

This report has focused on the core functionality and essential components of a platform that involves a large area of different technology, which generally leaves space for many future improvements and possible extensions of the system.

6.1.1 Implementation of Components

The basic functionality of the system has been implemented, considering essential classes and overall architecture. The *Storage Manager* component has been implemented using on a database management system, together with the *Verifier* component using BLAKE2 and ed25519. The robustness of both improvement can be improved.

The implementation of proof-of-work and collaborative filtering need to be improved significantly. Future work includes and implementation of a suitable memory-bound proof-of-work system, such as MTP-Argon2. Collaborative filtering has mainly been implemented for evaluation purposes. As part of future work, the rating system component should implement robust collaborative filtering, to supplement proof-of-work and make large scale attacks less effective.

The communication component has not been implemented.

6.1.2 Additional Components

Reputation System

A reputation system, examined in section 2.4.3, could be added to the system to derive peer reputation. Reputation can provide a different measure of trust, compared to rating similarity, which might be necessary to mitigate a range of attacks, e.g. to reduce effects of reusing attack profiles.

Content Search

A search component could be added to the system to allow users to search for content in the rating database. Metadata provides descriptions of content that could be used to discover content. The reviews could potentially be used to rank content according to its relevance to a user.

6.1.3 Explicit Trust

As described in section 2.1.2, explicit trust between users can be represented by allowing users to choose specific users they trust or does not trust. Because this is not ideal for all types of applications, it is not considered a core part of the platform. However, because explicit trust is very effective at avoiding attacks, it could be beneficial for some applications or users to have this feature. It should not be difficult to support and integrate it, potentially in combination with a reputation system.

6.1.4 Linked Metadata

Because metadata can identify or reference content using any type of identification scheme, it is possible that different metadata could reference the same content using different identifier types. E.g file content could be references using different types of hash functions.The platform could provide guidelines for types to motivate use of consistent references, but alternatively it should be considered to support a method for linking or combining metadata, within the platform.

This problem is similar to the issue of having near identical metadata, which could also be approached by having a mechanism to combine metadata.

Revisions

Another use of linking metadata is to support revisions. Different revisions of content, would currently be identified as separate content and separate metadata. It is relevant to consider if reviews of previous versions content, should be allowed to propagate to new versions. This has the benefit of allowing minor changes without loosing all existing reviews, however the risk of malicious changes is an essential issue. It should be examined if revisions can be supported in a safe way.

Bibliography

[1] Barabas, C., Narula, N., and Zuckerman, E., *Defending Internet Freedom through Decentralization: Back to the Future?*, MIT Digital Currency Initiative and the Center for Civic Media, Aug. 2017.

[2] Wang, L. and Kangasharju, J., "Measuring large-scale distributed systems: case of BitTorrent Mainline DHT", pp. 1–10, Sep. 2013, ISSN: 2161-3559. DOI: 10.1109/P2P.2013.6688697.

[3] Yli-Huumo, J., Ko, D., Choi, S., Park, S., and Smolander, K., "Where Is Current Research on Blockchain Technology?—A Systematic Review", *PLOS ONE*, vol. 11, no. 10, pp. 1–27, Oct. 2016. DOI: 10.1371/journal.pone.0163477.

[4] Jøsang, A., Ismail, R., and Boyd, C., "A Survey of Trust and Reputation Systems for Online Service Provision", *Decis. Support Syst.*, vol. 43, no. 2, pp. 618–644, Mar. 2007, ISSN: 0167-9236. DOI: 10.1016/j.dss.2005.05.019.

[5] Aggarwal, C. C., *Recommender Systems: The Textbook*, 1st. Springer Publishing Company, Incorporated, 2016, ISBN: 9783319296579.

[6] Douceur, J. R., "The Sybil Attack", in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, Springer-Verlag, 2002, pp. 251–260, ISBN: 3-540-44179-4.

[7] Burke, R., Mobasher, B., Williams, C., and Bhaumik, R., "Classification Features for Attack Detection in Collaborative Recommender Systems", in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2006, pp. 542–547, ISBN: 1-59593-339-5. DOI: 10.1145/1150402.1150465.

[8] Menezes, A. J., Vanstone, S. A., and Oorschot, P. C. V., *Handbook of Applied Cryptography*, 1st. CRC Press, Inc., 1996, ISBN: 0849385237.

[9] Tolia, N., Kozuch, M., Satyanarayanan, M., Karp, B., Bressoud, T., and Perrig, A., "Opportunistic Use of Content Addressable Storage for Distributed File Systems", in *In Proceedings of the 2003 USENIX Annual Technical Conference*, 2003, 127–140.

[10] Kamvar, S. D., Schlosser, M. T., and Garcia-Molina, H., "The Eigentrust Algorithm for Reputation Management in P2P Networks", in *Proceedings of the 12th International Conference on World Wide Web*, ACM, 2003, pp. 640–651, ISBN: 1-58113-680-3.

[11] Fan, X., Liu, L., Li, M., and Su, Z., "EigenTrust++: Attack resilient trust management", in *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, Oct. 2012, pp. 416–425, ISBN: 978-1-4673-2740-4.

[12] Xiong, L. and Liu, L., "PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities", *IEEE Trans. on Knowl. and Data Eng.*, vol. 16, no. 7, pp. 843–857, Jul. 2004, ISSN: 1041-4347. DOI: 10.1109/TKDE.2004.1318566.

[13] Mehta, B. and Nejdl, W., "Attack Resistant Collaborative Filtering", in *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '08, ACM, 2008, pp. 75–82, ISBN: 978-1-60558-164-4.

[14] Dwork, C. and Naor, M., "Pricing via Processing or Combatting Junk Mail", in *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, Springer-Verlag, 1993, pp. 139–147, ISBN: 3-540-57340-2.

[15] Back, A., "Hashcash - A Denial of Service Counter-Measure", Tech. Rep., 2002.

[16] Carvalho, C., "The Gap between Processor and Memory Speeds", 2002.

[17] Biryukov, A. and Khovratovich, D., "Egalitarian computing", *CoRR*, vol. abs/1606.03588, 2016. arXiv: 1606.03588.

[18] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D., "Epidemic Algorithms for Replicated Database Maintenance", in *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, 1987, pp. 1–12, ISBN: 0-89791-239-X. DOI: 10.1145/41840.41841.

[19] Ganesh, A. J., Kermarrec, A.-M., and Massoulié, L., "Peer-to-Peer Membership Management for Gossip-Based Protocols", *IEEE Trans. Comput.*, vol. 52, no. 2, pp. 139–149, Feb. 2003, ISSN: 0018-9340. DOI: 10.1109/TC.2003.1176982.

[20] Voulgaris, S., Gavidia, D., and Steen, M. van, "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays", *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–217, Jun. 2005, ISSN: 1573-7705. DOI: 10.1007/s10922-005-4441-x.

[21] Leitao, J., Pereira, J., and Rodrigues, L., "HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast", in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, IEEE Computer Society, 2007, pp. 419–429, ISBN: 0-7695-2855-4. DOI: 10.1109/DSN.2007.56.

[22] Saint-Andre, P. and Klensin, J., "Uniform Resource Names (URNs)", Internet Engineering Task Force (IETF), RFC 8141, Apr. 2017.

[23] "Information and documentation – Digital object identifier system", International Organization for Standardization, ISO 26324:2012, May 2012.

[24] "Secure Hash Standard (SHS)", National Institute of Standards and Technology, U.S. Department of Commerce, FIPS Publication 180-4, Aug. 2015. DOI: 10.6028/NIST.FIPS.180-4.

[25] "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", National Institute of Standards and Technology, U.S. Department of Commerce, FIPS Publication 202, Aug. 2015. DOI: 10.6028/NIST.FIPS.202.

[26] Aumasson, J.-P., Neves, S., Wilcox-O'Hearn, Z., and Winnerlein, C., "BLAKE2: Simpler, Smaller, Fast As MD5", in *Proceedings of the 11th International Conference on Applied Cryptography and Network Security*, Springer-Verlag, 2013, pp. 119–135, ISBN: 978-3-642-38979-5. DOI: 10.1007/978-3-642-38980-1_8.

[27] Bernstein, D. J. and Lange, T., *eBATS: ECRYPT Benchmarking of Asymmetric Systems*, Last accessed 29 May 2018, Feb. 2018. [Online]. Available: https://bench.cr.yp.to/results-sign.html.

[28] Gligoroski, D., Ødegård, R. S., Jensen, R. E., Perret, L., Faugère, J.-C., Knapskog, S. J., and Markovski, S., "MQQ-SIG: An Ultra-fast and Provably CMA Resistant Digital Signature Scheme", in *Proceedings of the Third International Conference on Trusted Systems*, Springer-Verlag, 2012, pp. 184–203, ISBN: 978-3-642-32297-6. DOI: 10.1007/978-3-642-32298-3_13.

[29] Bernstein, D. J., Duif, N., Lange, T., Schwabe, P., and Yang, B.-Y., "High-speed High-security Signatures", in *Proceedings of the 13th International Conference on Cryptographic Hardware and Embedded Systems*, Springer-Verlag, 2011, pp. 124–142, ISBN: 978-3-642-23950-2.

[30] Percival, C., *Stronger key derivation via sequential memory-hard functions*, 2009.

[31] Biryukov, A., Dinu, D., and Khovratovich, D., "Fast and Tradeoff-Resilient Memory-Hard Functions for Cryptocurrencies and Password Hashing.", *IACR Cryptology ePrint Archive*, vol. 2015, p. 430, 2015.

[32] Tromp, J., *Cuckoo Cycle: a memory bound graph-theoretic proof-of-work*, Jul. 2015.

[33] Biryukov, A. and Khovratovich, D., *Equihash: Asymmetric Proof-of-Work Based on the Generalized Birthday Problem*, Cryptology ePrint Archive, Report 2015/946, 2015.