An architecture for data integrity in untrustworthy social networks

Carlos Eduardo da Silva [*]
Sheffield Hallam University
Sheffield, UK
C.daSilva@shu.ac.uk

Angus Young
Sheffield Hallam University
Sheffield, UK
angus@dotwave.io

ABSTRACT

Social media platforms have the power to massively influence public opinion, and as such, are under increased pressure to submit to state regulation. They operate under a centralised model, in which users should trust the service provider that the information being presented has not been altered. This paper presents an architecture for enabling independent verification of content integrity in social networks. Our solution assumes a scenario of an untrustworthy social network provider, utilising public key cryptography for signing user content, and distributed hash tables for storing detached signatures. We have developed a proof of concept system that has been used to demonstrate the feasibility of the proposed solution.

CCS CONCEPTS

• **Information systems** → **Social networks**; • **Security and privacy** → **Social network security and privacy**;

KEYWORDS

Online social networkss, Security, Data integrity, Public-key cryptography, Distributed hash table

ACM Reference Format:
Carlos Eduardo da Silva and Angus Young. 2022. An architecture for data integrity in untrustworthy social networks. In *The 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22), April 25–29, 2022, Virtual Event, .* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3477314.3507001

1 INTRODUCTION

We live in a world where public discourse, and in some cases diplomacy are increasingly conducted on social media. In fact, it provides a platform for powerful figures to spread their message. In March of 2020, Twitter marked a video widely shared by the Trump administration of Democratic frontrunner Joe Biden appearing to tell Americans to vote for Trump as "significantly altered or fabricated" [15]. The same video continued to be circulated on Facebook, and has been viewed by millions of people. While the impact of this video on voting cannot be directly measured, the degree to which it was circulated highlights the potential of social media content, modified or otherwise to be propagated across the internet.

The companies that provide these platforms are coming under increased political pressure to submit to state regulation, largely in light of the 2018 Cambridge Analytica[1] scandal and the scale of the influence that major social media platforms (or Online Social Networks, OSN) were found to have had in the 2016 UK-EU independence referendum[2] to mention two examples. The form that regulation might take, or which organisations might implement it is still unclear.

If these platforms continue to operate from within black boxes, their potential to steer the global conversation will become even more problematic. Jack Dorsey, the CEO of Twitter spoke about his own scepticism of large technology companies:

> [[2]] *We have aspirations to serve every person on the planet ... we have to think deeply about how we might distribute and decentralise this work. I have a lot of scepticism of companies like ours and leaders like me.*

It is clear that social media platforms have the power to massively influence public opinion. Whether they actively set out to do so is another question; if they did, how would you know? All of the major platforms are highly centralised, and without third party oversight there is no way of knowing that the information they are presenting has been altered by a malicious actor or the platform itself. For example, some governments enforce laws that prohibit online services from operating in their countries if the data is not stored domestically (See [9]). These governments could theoretically gain access to this data and modify it without the knowledge of the provider or the end user. On a smaller scale, many businesses use internal social media-like applications for internal and inter-company communication. These networks also have the potential to be compromised and manipulated by an unauthorized party.

Even with regulation, data centralisation still leaves the company that owns the data with the ability to change whatever data they want opaquely [6]. In a traditional centralised social media platform (OSN), the burden of trust is placed on the provider. Users trust that the provider will ensure that their data is secure, and that other users cannot modify it without their permission.

In this context we conducted an exercise where this assumption is removed and the provider is assumed to be untrustworthy, or unable to maintain the integrity of user data (for example, in the case that a 3rd party has access to their servers). Thus, this paper presents

[*]Corresponding Author.

[1]https://www.vox.com/policy-and-politics/2018/3/23/17151916/facebook-cambridge-analytica-trump-diagram.
[2]https://www.ox.ac.uk/news-and-events/oxford-and-brexit/brexit-analysis/views-from-oxford

an approach for verifying that content served by a OSN platform is identical to content supplied to it. We designed an architecture that provides third parties with the ability to verify content and its author independently from the OSN service provider. A proof of concept has been developed employing public key cryptography for achieving content integrity and distributed hash tables (DHT) [21] for storing signatures. In this way compromising the OSN opaquely would require collusion from multiple parties, and by increasing the number of involved parties the system would be harder to compromise.

Our proof-of-concept has been implemented following the same microblogging format of Twitter, employing cryptography software PGP [4] and the Kademlia DHT implementation (the same used for bittorrent), being able to provide an user a clear indication when content has been modified by someone other than its author. This proof-of-concept has then been evaluated in terms of its performance and effectiveness based on the STRIDE threat model. The results obtained with this proof-of-concept demonstrated the capability of verifying the integrity of social media content independently of the OSN itself.

This paper is organised as follows: Section 2 discuss related work and existing solutions. Section 3 presents the design of the proposed system and a definition of its scope and functionality. Section 4 covers the system's implementation, deployment and any issues faced. Section 5 presents an evaluation of our approach, testing of the DHT and a threat model. Section 6 concludes the paper with a brief discussion on its limitations and avenues for potential future work.

2 RELATED WORK

Online Social Networks (OSN) have received considerable attention due to several issues uncovered throughout the time, motivating the creation of Decentralised Online Social Networks (DOSN) [6], i.e., an OSN implemented on a distributed platform. [6] have performed a comprehensive survey on DOSN analysing several works that explore DHT to store the social content or as an indexing service, such as, PeerSon [1] and My3 [13]. They also identify real deployments of DOSN, for example Diaspora[3] and Mastodon [16], which are not fully decentralised as they work based on a federation of trusted servers independently operated. Our work takes inspiration in these approaches. However, the main difference is that we are not trying to recreate the whole OSN, but recognising the existence of an underlying OSN platform, while exploring the DHT to store content signature.

More recently there has been a movement on exploring Distributed Ledger Technologies (DLTs) for DOSN [5]. DLT are append-only distributed records, where new data is verified by peers using a consensus algorithm. Amongst the existing approaches we can find works that saves all posts and social interaction information in a DLT, such as SteemIt [10]. There are also some approaches that employ some sort of distributed storage system for OSN content, such as DHT-based InterPlanetary File System (IPFS)[4] where post contents are saved (encrypted or not), and a DLT for storing social interaction information along-side a pointer to the actual content [7, 20].

In fact, we have considered DLT as an initial solution to our approach. However, the problems identified by [5] with the use of DLT for OSN, such as the lack of identity checking capabilities and scalability issues, have made us reconsider our approach. Another issue with the suitability of DLTs for use within OSNs is the inability for data to be removed from the ledger. This is incompatible with some regulations on data storage such as GDPR's "Right to erasure", which specifies that upon request from a user, a company must be able to completely remove all records that they hold in regards to that user [3, p. 43, article 17]. A potential solution to this problem involves encrypting all user data, and then "losing" the cryptographic key to this content in order to render it unreadable, although this raises issues regarding future-proofing encryption procedures.

3 SOLUTION DESIGN

Figure 1 shows a high level overview of the applications involved in the system and the operations they perform. Central to the operation of our proposal is the client application, which performs the signing and verification of content. The client application corresponds to a front-end Single Page Application commonly used by Twitter and Facebook. Private and public keys are stored locally by the client, which provides standard life-cycle operations for key management and capabilities for creation and signing of plain text content by a user, and verification of content by another independent user.

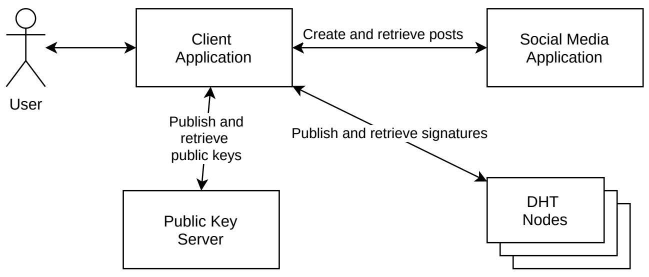

User
Client Application
Create and retrieve posts
Social Media Application
Publish and retrieve public keys
Publish and retrieve signatures
Public Key Server
DHT Nodes

Figure 1: General overview of the proposed solution.

To ensure content integrity, our solution uses PGP detached signatures to sign content [4]. These signatures will be stored in a public distributed hash table (DHT). Distributed hash tables (DHT) are commonly used in distributed file system applications, peer to peer (P2P) networking and web caching. It operates similarly to the traditional hash table data structure, whereby a hash function is used to calculate an associative key for any given value, except the data is distributed among multiple nodes [21]. Third parties can participate in the table by bootstrapping their nodes from one or more static nodes. In this way, a client can use any node of the DHT to perform a lookup for a signature, and then verify post content against a public key retrieved from a public key server (PKS). Public key servers (or PKS) provide an interface for publishing and retrieving public keys over HTTP. Users wishing to verify or encrypt content can use these servers to search for keys by email address or by key fingerprints.

[3]https://joindiaspora.com/
[4]https://ipfs.io

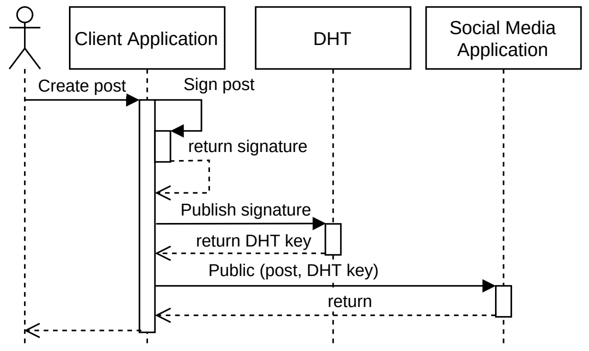


Figure 2: Sequence diagram describing the post creation flow.

Figure 2 shows the process of a user creating a post. Once the user creates some plain text content the client generates a detached PGP signature for this content using their private key, and publishes it to a DHT node via an API call. This API call will return the status of the insert operation. If it succeeds, it will also return a SHA1 hash of the inserted content, which will act as a DHT key to retrieve this value later. This key, along with the plain text post content, are published to the OSN via another API call.

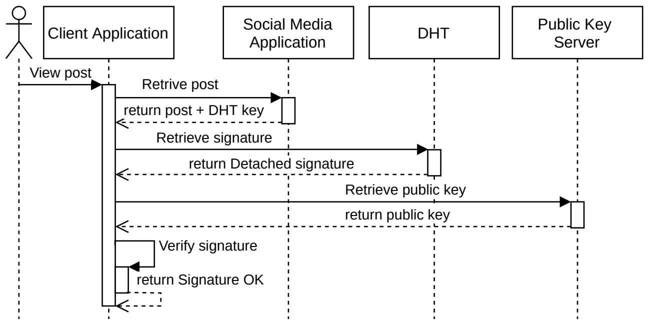


Figure 3: Sequence diagram describing the post verification flow.

Figure 3 shows the process of a user verifying a post. As a prerequisite, the user must have published a public key to the PKS. This public key must be associated with the email used to access the OSN, and it must have been verified. The client application receives a post from the OSN, containing the post's content, its author and a DHT key. The user retrieves the post's signature from the DHT via an API call. This signature is then verified against the authors public key, retrieved from the PKS. If the verification is successful, the post is considered to be verified.

4 SYSTEM IMPLEMENTATION

The system architecture is outlined in Figure 4. This figure gives a low-level overview of the applications that make up the system, and defines the boundaries of 1st party (the OSN provider) and 3rd party deployments. This extends the system overview presented in Figure 1. Each of the following components follow a microservice architecture, and aim to operate completely independently of each other.

The *Social Media Platform* is designed to act as a very basic online social network (OSN), to demonstrate how this system design could be applied to existing social media applications. It follows the same microblogging format implemented by Twitter[5], with users posting short plain text posts and viewing the posts of others in a chronological "feed". This format was chosen as it is familiar to most users of OSNs. It is generic in function, and could operate completely independently to the verification and signature publishing services that the client facilitates and interacts with.

The social media platform is written in Java, and serves HTTP requests via RESTful endpoints provided by the Spring framework (*REST Interface* component). The application utilises Spring's inbuilt *message broker* to provide websocket endpoints for clients to consume messages from. Post and user data is stored in a *Mongo database* via Spring Data's ORM (Object Relational Mapping) implementation for Mongo.

When identifying a DHT implementation for use within this system, the open arrival and departure of nodes was essential, in order to enable third parties to participate. For this reason, we chose Kademlia [12], the DHT system underlying BitTorrent, which operates the largest DHT in the world with over 10m nodes participating daily [18]. The DHT nodes have been implemented using the Bittorrent DHT implementation in Javascript[6]. It allows for bootstrapping (initialising the node with an exiting hash table from another node), and provides an interface for inserting and retrieving values from the table via an abstraction of the protocol's kRPC interface [11].

To utilise this functionality in the system, each node needed to expose this functionality over the Web. To accomplish this, the insert and retrieve functions were encapsulated by an API written in Node.js using the Express framework[7], providing a RESTful API for interacting with the DHT. By exposing these endpoints on each node, any node in the table can be used as an entry point, minimising points of failure. We assume that the bootstrap node is maintained by the service provider, considered as an static DHT node, and that 3rd party nodes would be available, as is the case in the Bittorrent network. This means that the availability of signatures is directly linked to the availability of nodes within the hash table. Depending on the number of posts, *n* nodes must be available for all signatures to be retrieved.

The client is written in Vue.js[8], a JavaScript framework designed for building reactive web applications. The client needed to be transparent enough in its operation that should the user decide that they do not trust the client, they could replicate any functionality, or re-implement the client in its entirety. The client has 5 core functions: Displaying post content; Managing keypairs (generating and publishing new keypairs, revoking pre-existing keypairs, importing and exporting keypairs); Verifying post content; Signing posts; and Managing user state.

The signing and verification operations have been implemented using OpenPGP.js library, a javascript implementation of the OpenPGP protocol, providing an API for PGP cryptography within the browser. The client's interactions with the OSN are RESTful calls for authentication, retrieving and creating posts, and a websocket connection

[5]https://twitter.com
[6]Bittorrent-dht v9.0.3, https://github.com/webtorrent/bittorrent-dht
[7]Express v4.17.1, https://expressjs.com/
[8]https://vuejs.org/

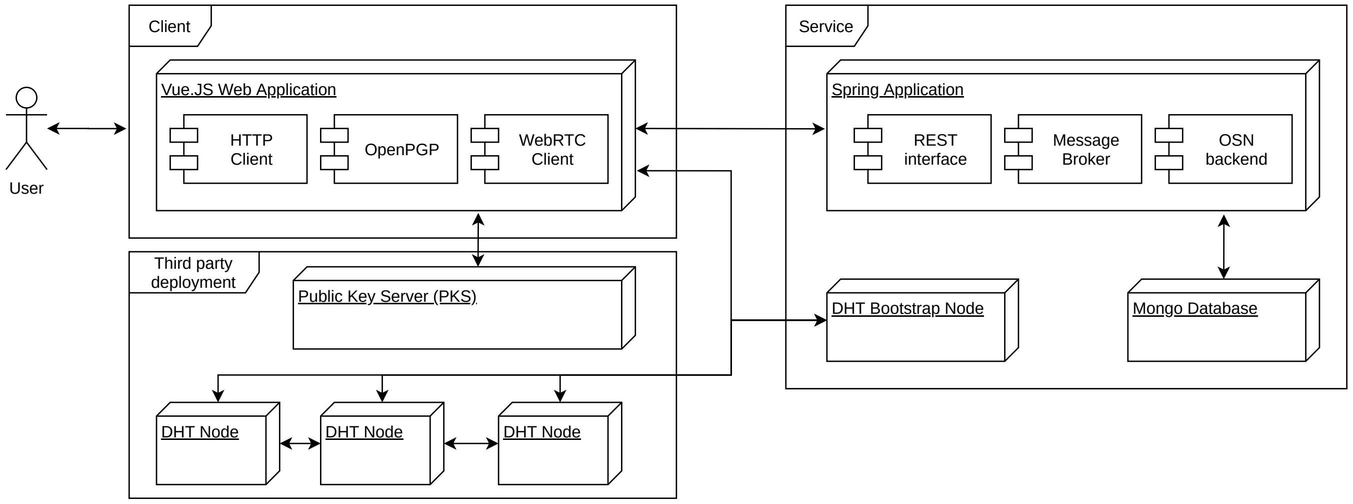
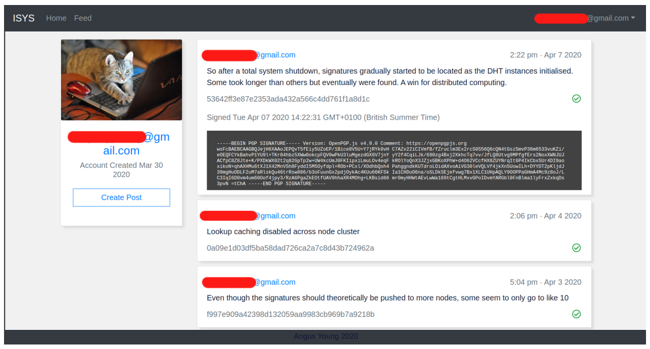


Figure 4: Deployment diagram of the system architecture implementation.

to the message broker endpoint for displaying new posts in real-time. It interacts with the DHT API and the PKS API to retrieve public keys and publish signatures to the DHT. In a production environment, the user should be able to select providers for these services, in the case that one particular provider is not trusted, and to reduce the number of points of failure.

State management in Vue is handled using the Vuex library. Vuex uses the concept of 'stores' to manage stateful data. Within stores, methods for accessing and modifying (mutating) data are separated in order to enforce access rules. In the case of the client application, Vuex maintains stores for the currently logged in user (profile data, authentication tokens), and keypairs. Keypairs are persisted in application storage after a user logs out, and then loaded into keypair store when the user logs back in.


ISYS Home Feed [redacted]@gmail.com

[redacted]@gmail.com
Account Created Mar 30 2020
Create Post

[redacted]@gmail.com 2:22 pm · Apr 7 2020

So after a total system shutdown, signatures gradually started to be located as the DHT instances initialised. Some took longer than others but eventually were found. A win for distributed computing.

53642ff3e87e2353ada432a566c4dd761f1a8d1c ✓

Signed Tue Apr 07 2020 14:22:31 GMT+0100 (British Summer Time)

```
-----BEGIN PGP SIGNATURE----- Version: OpenPGP.js v4.9.0 Comment: https://openpgpjs.org
wsFcBAEBCAAGBQJejH6XAAoJEPQvT5fEiy5UZoEP/1B1ce8V5U+Y7jRYk0vH C7AZv221CIVmfB/fZruclm3Ex2rcS0S56Q6cQN4tGszSmvP36m6533vuKZ1/
eOEQFCYkBahvP1YU9l+TKr04hbz5XWw0okcpFQV0wPkU3luMgezdGX6V7jnY yY2f4CqiLJk/690zg4Bxj2XkhcTq7vv/JfLQ8Utvg9MPfgfErs2NoxXWNJUJ
ACfpC8Z8Jte+K/PXDkWX02t2q82GpTp2w+UW4kcUmJ0FKI1pxiLmuLOv4eqF kROlYoQnX3JZjxGBKoXPhW+d4062VCcfHX8ZUYNrqIt6P4IkCbxSUr4DI9ao
xikoN+qhAXHMu6tXJ1X42MnVSh8FyddI5M5Oyfdpl+ROb+PCxl/XOdhbQoh4 PahggndkKUTdroLOidAXvoAiVG30leVQLVf4jkXn5UowILh+DYYDT2pKljdJ
39mgHuODLF2uM7aRlskQu46trRsw886/b3oFuunGx2pdjOykAc4KUu66KFSk Ia1CHDuO6na/oSLDkSEjefvwg7Bx1XLC1UHpAQLY9OOPPaGHmA4Mc9z8oJ/L
C3IqI6D0vm4um60Oof4jpy3/RzA6PgaZkEOtfUAV9hhaXR4MOhg+LKBsid66 mr0myHHWtAEvLwWa169tCgtHLMxvGPoIDvmYARGbl0FnBlma1lyFrxZxkqDs
3pvN =tChA -----END PGP SIGNATURE-----
```

[redacted]@gmail.com 2:06 pm · Apr 4 2020

Lookup caching disabled across node cluster

0a09e1d03df5ba58dad726ca2a7c8d43b724962a ✓

[redacted]@gmail.com 5:04 pm · Apr 3 2020

Even though the signatures should theoretically be pushed to more nodes, some seem to only go to like 10

f997e909a42398d132059aa9983cb969b7a9218b ✓

Angus Young 2020

Figure 5: Live view of the client feed page.

The client has two main pages, one for viewing a list of chronologically ordered posts (/feed, see figure 5), and another for managing the currently logged in user's keys. Posts displayed on the feed include a status indicator for the verification state of the post. A verified post will display a green tick, which can be clicked to reveal the signature that was retrieved from the DHT. A post that fails verification or that doesn't have an associated DHT signature key will display a red error symbol.

The key management page allows a user to generate a new keypair, of which the public key is automatically published to the PKS, import an existing keypair, or export the currently loaded keypair as a zip archive. The client also has a basic profile page (/profile/username) that displays the users public key and a history of their posts.

Mailvelope's public key server was used to create a sandboxed PKS instance for the system to interact with. This keyserver software was chosen because it has a well-documented RESTful API, and is completely open source. The PKS had to be sandboxed to avoid polluting the public keypool with testing signatures. To integrate this keyserver with the system, it needed to be containerised to run alongside the other applications. To do this the main repository was forked and a Dockerfile container configuration was created.

Each of the above applications is built and deployed as a container image running on a private container registry hosted on a virtual private server (VPS). Portainer, a web dashboard for managing a Docker instances, is then used to deploy these container images.

To orchestrate deployment, docker-compose was used to create "stacks" (a group of containers that share a network) of containers. Figure 6 shows an overview of the containers deployed within the system and the flow of Web traffic.

We have deployed out client application in an isolated container, acting as the frontend for the whole system. Two stacks comprise the backend system; The first stack represents the provider's domain, and is composed of a single instance of the OSN API and its MongoDB instance. The second stack represents external (or third party) services, such as the PKS (PKS API) and its database.

Three instances of the hash table container image are also managed by this stack (DHT Node). Multiple instances were used so if one instance goes down, the routing table is maintained by the others. Each of these hash table nodes bootstrap off of each other, so if a fatal event occurred and a container had to restart, it would

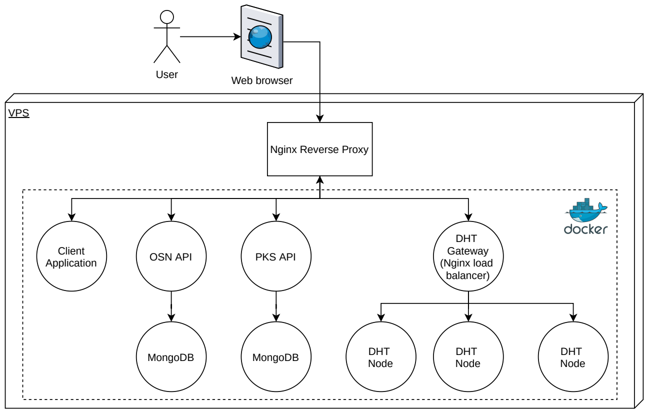


Figure 6: Overview of the container stacks used for deploying our proof-of-concept application.

automatically recover its state. To distribute bootstrap requests and lookups to the hash table, a Nginx[9] container instance acting as a gateway provides load balancing. All access to these containers is routed through a nginx reverse proxy running on the VPS.

5 EVALUATION

This section presents an evaluation of our approach. We have conducted a series of tests to demonstrate its effectiveness, performed a preliminary performance evaluation and a threat analysis based on the STRIDE threat model [17].

5.1 Solution Effectiveness

We initially considered the goals of the system against the system’s implementation. The Java API was tested using the JUnit testing framework, which provides a harness for writing unit and integration tests. Tests were written using a mock client provided by Spring, which makes requests to the API running in a test environment managed by JUnit. These tests ensure that the API forms responses and errors as expected, and that changes made to the API do not cause regressions in functionality or any deviations from spec.

After that we have devised different scenarios and conducted tests based on them. These scenarios consider two users (user A and user B) with their respective keypairs created and registered within our client application, and public key published into the PKS.

- **Uncorrupted Post**: This scenario considers the situation in which one user signs a post and another user successfully validates it. User A creates some plain text content and signs it using their private key. The signature is then successfully published to the hash table addressed via the bootstrap node gateway. The DHT key returned from this operation, along with the plain text post content is published to the OSN. User B retrieves this content from the OSN, and performs a lookup for its corresponding signature in the DHT. User B then verifies the signature against the content and User A’s public key. The verification is confirmed manually via the gpg command line tool.
- **Corrupted Post**: This scenario involves the content of the post being modified by a party with access to the database (such as the OSN provider). As above, user A signs and posts some content to the platform, and publishes the post signature to the hash table. The post is then manually modified in the database. User B then attempts and fails to verify the message. The client reports that the content is corrupted. The corruption is again confirmed using the gpg command line tool.
- **Corrupted Signature**: In this scenario, it is assumed that a node n is malicious or operating out of specification, and is incorrectly storing signatures in such a way that it indicates to peers that it has the data, and returns an invalid or corrupted signature. User A signs and posts content, and the signature is successfully stored in the DHT. When user B retrieves the signature from the hash table, the malicious node responds with the corrupted key. On attempting validation, the client of user B reports that the signature is invalid, or that it does not match the public key associated with the post.

5.2 Hash Table Performance

Performance of DHT implementations have been extensively explored in the literature considering both intensive churn (node departure/arrival) and lookup intensive workloads (e.g., [14]). For example, [8] evaluates the performance of 5 popular DHT implementations (Chord, Kademlia, Kelips, OneHop. and Tapestry), and states that Kademlia (the underlying DHT implementation that Bittorrent-DHT is based on) suffers from longer lookup times under churn but lower overall bandwidth usage, due to its method of simultaneous peer discovery and lookup. It also states that Kademlia’s lookup times do not benefit from routing table stabilisation like other DHT implementations. For this reason, utilising static nodes as is done in this system is only beneficial for the purpose of guaranteeing available nodes for parallel lookups. Although churn performance is an important metric for evaluating the performance of any hash table based application, in the case of the proof of concept, lookup performance is more relevant, since the DHT is likely to have a high number of static nodes, and lookups are frequent.

In terms of lookup performance, Kademlia contacts $O(\log(n))$ nodes during lookups [12]. Therefore, as the number of nodes increase, we should expect to see a logarithmic increase in lookup time. Nevertheless we have conducted some lookup related experiments, as our approach’s use of Bittorrents DHT implementation is not typical of its designed use case.

To test the performance of the DHT in the context of our system, a test environment was created within Node.js v12.16.1 running on an i5-7200U with 16GB of RAM. The test environment takes two variables, n and c, which represent the number of hash table nodes and the number of values to insert respectively. A hash table node is a javascript object handled by Node.js event loop asynchronously. Each value is 858 characters long, the same length as a PGP detached signature. These values are random, and created

[9]Nginx is an open source web server, API gateway and load balancer. See https://www.nginx.com/.

using the crypto library. Insert and lookup operations are initiated from a random node within the environment. Varying values of n and c were supplied to the environment, focusing on the lookup failure count, which provides the basis for calculating the retention rate r of the DHT as the relation between the number of values inserted (c) and the number of lookup failures obtained.

It quickly became apparent that for values of $n < 30$, the DHT was unable to retain more than 1000 values. This was observed consistently with c varying between 2000 and 10000. With $n > 30$ the retention rate (r) became more inconsistent, for example, at $n = 80$ and $c = 5000$ the retention rate was 75.7%. At $n = 150$ and $c = 5000$ the retention rate increased to 98.9%. By considering $k = c/n$ the ratio between the number of values c and the number of nodes n it was observed that the retention rate was consistently at $r \geq 99\%$ for values of $k < 30$. This ratio is likely to be specific to the testing system.

Using this estimation, the average lookup time will be measured for n with $c = 26n$. Lookup time was measured for $n = 10$, 50, 100, 500, 1000 and 5000. A graph of these results should show a consistent increase in lookup time. It is worth noting that the resources available to the testing system were definitely a factor in performance; at values of $n \geq 5000$, the process started consuming most of the available memory, to the point where tests of performance at this scale could be considered inaccurate due to the system attempting to maintain operating system stability.

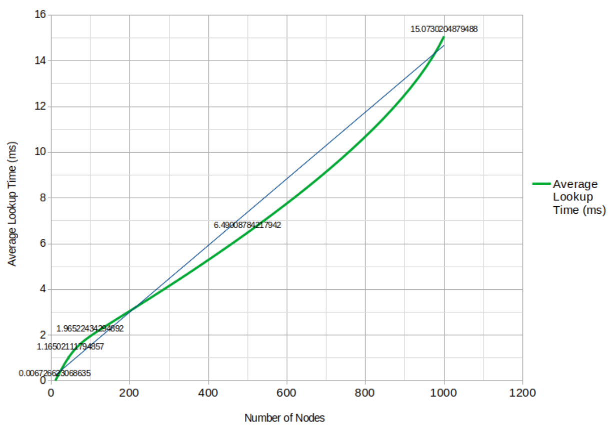


Figure 7: Lookup Time vs Number of Nodes

Figure 7 presents the results considering a 3 run average. As predicted, the line of best fit shows that average lookup times increased at the expected rate, as the number of participating nodes in the DHT increased. At values of $n = 1000$, an average lookup failure count of 227 (or 0.87%) was observed, demonstrating that the previously discussed ratio of nodes to values is not definitive, and that the testing system was a limiting factor in result accuracy. For the purposes of this experiment, values of $r \geq 99\%$ were considered to be within the margin of error.

For the OSN to operate on a similar scale to Mastodon [16], which as of October 2021 had 1.025M active users[10], and assuming each user creates at least two posts a day, and that the above ratio of nodes to posts holds true on other systems, the system would require at least 41,000 DHT nodes to be available to allow all signatures to be retrieved for a single day. If the availability of nodes was the same as in Bittorrents network, which as previously mentioned has around 10 million nodes participating daily, this would be feasible, but signatures that weren't retrieved often would eventually be evicted.

Due to the limitations of the testing system, more extensive testing and further exploration of DHT configuration parameters are neccessary to evaluate conclusively the performance of the DHT under this kind of load, as well as its behavior in terms of value retention.

5.3 Threat Model

The goal of the system is to prevent a malicious actor from changing a user's content without it being transparent that they have done so. They should also be unable to forge content to make it appear to have come from another user. We have used the STRIDE threat modelling [17] as basis to evaluate our approach in relation to these problems. STRIDE was chosen over alternative threat modelling techniques because it explicitly covers tampering and spoofing, security threats which this system aims to deal with.

For the purpose of this threat model, we are evaluating the security of the first party systems implemented. The security of the public key server will not be covered.

Spoofing. Since the ability of each user to sign posts is dependant on being able to decrypt their private key, to successfully spoof a user, an attacker would need: a user's account password; their private key; and the decryption passphrase of their private key.

In the case of our proof-of-concept, passwords are stored on the server as Bcrypt hashes, in line with best practices adopted by OSN providers. The most likely attack vector for accessing a user's account password would be a phishing attack, in which the user unwittingly provides a 3rd party with their credentials. Once they have access to this password, **the attacker can create, but not sign posts**. To other users, any posts created by the attacker would not be marked as verified. For this reason, at this stage the attacker could not be considered to have spoofed a user.

To be able to sign posts, the attacker would first need to have access to the users private key. The easiest way for them to do this would be to have access to the users browser. On shared machines, this is trivial, but on remote machines this is far more complex, and subject to the expansive and well tested security measures implemented by operating systems and browsers. In the case that the attacker did gain access, the key would still need to be decrypted using the users decryption password. At this point, the security of the private key is dependant on the strength of the password used to encrypt it. Brute force attacks are possible, but unrealistic with a sufficiently complex password.

In summary, the worst case scenario for a spoofing attack is a user that uses the same, simple password for both their account

[10]Data obtained on 15/10/2021 from https://fediverse.party/en/mastodon

login and their private key, and uses the system via an unsecured or public browser. It is also worth noting that the same security principles applied to handling and storing private keys must also be adopted by the user, who could willingly or unwillingly provide their private key (encrypted or otherwise) to a third party, and circumvent the security measures adopted by this system.

Tampering. Regarding data in transit, all communication is based on the HTTPS protocol, meaning content is encrypted from the server to the client. Thus, in a situation of Man in the Middle Attack (MITM) the attacker could make the social media platform believe that the request came from the user, and have it store manipulated data. However, other clients would be unable to verify this content, with or without the social media platform providing a signature key.

Regarding tampering of data within the platform, we consider that OSN databases are hosted on a private network with limited connectivity. If an attacker had access to the server however, they would potentially be able to manually modify the database. This also means that the provider has the potential to modify data. Just as mentioned previously, any modification to users posts would result in verification failing on the client.

Regarding tampering of data within the Distributed Hash Table, due to the nature of how hash tables function, data is immutable once inserted. It is however possible in the Bittorrent DHT implementation to modify a cached value on a node, which it will return to the user instead of performing a lookup from other nodes. Although this behaviour can be overridden to always perform the lookup, the potential still exists for modified cached values to be returned from a rogue node. This doesn't pose much of an issue to the system in general, as the bootstrap node that most clients will query will always perform the lookup. In an production environment, the client should allow users to choose their entrance node in order to balance power between nodes, and to avoid the bootstrap node(s) from becoming points of failure or attack.

The client application could be used as a vector for attack. If a malicious party managed to alter the client application they could present unverified content as verified, or divert calls from the real hash table and public key servers to their own, which could return data that make content appear valid. The system alone cannot circumvent this, but the availability of public key servers means that if a user was to make their own query to the PKS, they could find that the public key presented by the client is incorrect, and thus reveal the tampering. There is also the potential for a malicious party to prevent a user from signing posts by modifying or removing their private key (in browser storage). If the user has not exported their private key and the one loaded into the web application was their only copy, they would effectively be unable to sign posts.

Repudiation. A user that has signed a post, then revoked that signature could effectively deny that they had signed it. This could be disproven if a copy of their previous public key was available, and it was confirmed to have been verified, however repudiation is not the primary goal of the system.

An attack from outside the system would be evident from container and system logs, however an attacker with access to the system could remove any trace of an attack by deleting or redacting logs. To mitigate this, a log aggregator such as ELK stack[11] could collate logs on an external system to reduce the likelihood of an attacker being able to hide their tracks.

Information Disclosure. No traffic to or from the bootstrap nodes, the client application or the platform itself is plain text. All traffic is sent via HTTPS, using a certificate signed by Cloudflare[12]. Traffic between the gateway (nginx) and the services is over HTTP, but never leaves the server. User passwords are stored as Bcrypt hashes in the database, but it would be possible for these to be logged by the platform as it processes the request, and so the potential exists here for passwords to be leaked. As mentioned previously, this doesn't threaten the primary goals of the system, but does mean that anyone with access to the service configuration could enable trace logging for the authentication controller and view plain text logs of incoming requests, which may contain passwords. This could be partially negated by implementing a log masker, but this would not be effective at hiding these logs from an attacker with insider access to the system (such as the platform provider). As configured, only warnings are logged by the platform, which do not contain request data.

Denial of Service. As deployed as a proof of concept, the social media platform itself is not configured to load balance requests or restrict traffic in the case of resource starvation. Since the system is deployed as containers, little extra configuration is required to redeploy the system within a Kubernetes cluster or behind an nginx load balancer to provide this functionality. This means that in its current state, the platform is vulnerable to flooding attacks and DDOS.

The 3 DHT nodes deployed to facilitate third party bootstrapping operate behind an nginx load balancer, but since they operate on the same system behind the same network interface they are also susceptible to flooding/DDOS attacks.

To negate the risk of DOS attacks in production, each microservice (or cluster of microservice instances) could be deployed on independent systems and accessed via an API gateway that facilitates load balancing and request routing.

Elevation of Privileges. The system is deployed as a set of microservices within containers managed by Docker. These containers run on a VPS based on a long term support version of Ubuntu, which regularly gets security patches. Container groups (or stacks) are stopped and started via Portainer, a Web management dashboard for Docker. This dashboard requires a username and password combination for access. Portainer supports role based access control, and the user that can manage the containers within the system's stack cannot manage other containers or create new ones. The administrator user can only authenticate from the server itself, and so cannot be used remotely. If a user was to bypass these security measures, they would also gain shell access to each container instance. In this case, an attacker would be able to modify any data they wished, or take the system down. This could be mitigated by incorporating two factor authentication into the Portainer.

[11]https://www.elastic.co/elastic-stack/

[12]Cloudflare is a web services company that provides DNS and CA services. See https://www.cloudflare.com/

5.4 Discussion

In conclusion, our POC system is capable of preventing attacks that aim to spoof user content through its use of detached signatures. This is especially true when the provider of the OSN is considered to be a threat. We would still be susceptible to data tampering, mainly when considering the client application as attack vector. However this scenario would also be true of any OSN platform, and mitigation techniques already employed could be reused in our system.

Our system using Kademlia Bittorrent DHT implementation demonstrates the feasibility of our approach regarding the number of nodes required and the number of nodes available. In terms of a proof-of-concept we consider these to be very positive results, in line with the existing body of work on DHT performance. More performance related experiments are needed, mainly considering that we used standard Bittorrent parameters for the DHT configuration. This is also aligned with other solutions for descentralised/federated OSN such as peertube[13], which employes Webtorrent for storing videos, a DHT implementation compatible with Bittorrent.

Another aspect that deserves further investigation is the use of PGP and public key server to handle user's keys. That adds a number of requirements that are difficult to implement in large scale, mainly related to public key distribution. It is important to mention that PGP and PKS were used as a proof-of concept due to their simplicity of implementation. Those can, and should, be replaced by more appropriate public key distribution approaches, such as the mechanisms used in some instant messages application, such as Signal and more recently Matrix[14].

6 CONCLUSION

This paper presented an architecture for ensuring integrity of user content in Online Social Network (OSN), using a combination of distributed computing and public key cryptography. User posts are signed using PGP detached signatures that are then stored in a DHT. We developed a proof-of-concept[15] to demonstrate the effectiveness of our solution employing the Bittorrent-dht implementation for storing content signature.

As our solution is a POC, it doesn't cover some of the traditional features of OSNs, such as user to user messenging, post commenting or multimedia content. Thus, an obvious direction of future work would be extending the functionality of the POC to provide more of the features available in traditional OSNs, while maintaining a focus on content integrity. We are currently investigating the ActivityPub [19] protocol used in the Mastodon OSN with the objective of integrating our solution to one of their existent client.

An obvious future direction involves dealing with the key distribution problem of using PGP, maybe exploring mechanisms to facilitate key exchange, such as the approach adopted by the Matrix system. It is also necessary to deal with privacy related issues, which have been considered out-of-scope for the moment. Another future work is to further explore DHT implementations and configurations. The Bittorrent DHT implementation was designed for use within torrenting, a use case in which there is a high level of churn (arrival/departure of nodes), and persisting data is not a priority.

[13]https://joinpeertube.org
[14]https://matrix.org/
[15]The POC is available on demand by contacting the main author.

REFERENCES

[1] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. 2009. PeerSoN: P2P Social Networking: Early Experiences and Insights. In *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems* (Nuremberg, Germany) *(SNS '09)*. Association for Computing Machinery, New York, NY, USA, 46–52. https://doi.org/10.1145/1578002.1578010

[2] Jack Dorsey. [n.d.]. *The Joe Rogan Experience.* https://www.youtube.com/watch?v=_mP9OmOFxc4

[3] Council of the European Union European Parliment. [n.d.]. *Regulation on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (Data Protection Directive).* https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679&from=EN

[4] Hal Finney, Lutz Donnerhacke, Jon Callas, Rodney L. Thayer, and David Shaw. [n.d.]. *OpenPGP Message Format.* https://doi.org/10.17487/RFC4880

[5] Barbara Guidi. 2020. When Blockchain meets Online Social Networks. *Pervasive and Mobile Computing* 62 (2020), 101131. https://doi.org/10.1016/j.pmcj.2020.101131

[6] Barbara Guidi, Marco Conti, Andrea Passarella, and Laura Ricci. 2018. Managing social contents in Decentralized Online Social Networks: A survey. *Online Social Networks and Media* 7 (sep 2018), 12–29. https://doi.org/10.1016/j.osnem.2018.07.001

[7] Le Jiang and Xinglin Zhang. 2019. BCOSN: A Blockchain-Based Decentralized Online Social Network. *IEEE Transactions on Computational Social Systems* 6, 6 (dec 2019), 1454–1466. https://doi.org/10.1109/tcss.2019.2941650

[8] Jinyang Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil. 2005. A performance vs. cost framework for evaluating DHT design tradeoffs under churn. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, Vol. 1. 225–236 vol. 1. https://doi.org/10.1109/INFCOM.2005.1497894

[9] KPMG. [n.d.]. *The "localisation" of Russian citizens' personal data Compliance with the Russian law on personal data.* https://home.kpmg/be/en/home/insights/2018/09/the-localisation-of-russian-citizens-personal-data.html

[10] Chao Li and Balaji Palanisamy. 2019. Incentivized Blockchain-Based Social Media Platforms: A Case Study of Steemit. In *Proceedings of the 10th ACM Conference on Web Science* (Boston, Massachusetts, USA) *(WebSci '19)*. Association for Computing Machinery, New York, NY, USA, 145–154. https://doi.org/10.1145/3292522.3326041

[11] Andrew Loewenstern and Arvid Norberg. [n.d.]. *DHT Protocol.* http://www.bittorrent.org/beps/bep_0005.html

[12] Petar Maymounkov and David Mazières. 2002. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Peer-to-Peer Systems*, Peter Druschel, Frans Kaashoek, and Antony Rowstron (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 53–65.

[13] R. Narendula, T. G. Papaioannou, and K. Aberer. 2012. A Decentralized Online Social Network with Efficient User-Driven Replication. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing.* 166–175. https://doi.org/10.1109/SocialCom-PASSAT.2012.127

[14] Zhonghong Ou, Erkki Harjula, Otso Kassinen, and Mika Ylianttila. 2010. Performance evaluation of a Kademlia-based communication-oriented P2P system under churn. *Computer Networks* 54, 5 (apr 2010), 689–705. https://doi.org/10.1016/j.comnet.2009.09.022

[15] Martin Pengelly. 2020. Trump retweets video of Biden labelled by Twitter as 'manipulated media'. *The Guardian* (2020). https://www.theguardian.com/us-news/2020/mar/09/trump-retweets-video-manipulated-media

[16] Aravindh Raman, Sagar Joglekar, Emiliano De Cristofaro, Nishanth Sastry, and Gareth Tyson. 2019. Challenges in the Decentralised Web: The Mastodon Case. In *Proceedings of the Internet Measurement Conference* (Amsterdam, Netherlands) *(IMC '19)*. ACM, 217–229. https://doi.org/10.1145/3355369.3355572

[17] Riccardo Scandariato, Kim Wuyts, and Wouter Joosen. 2015. A Descriptive Study of Microsoft's Threat Modeling Technique. *Requir. Eng.* 20, 2 (June 2015), 163–180. https://doi.org/10.1007/s00766-013-0195-2

[18] L. Wang and J. Kangasharju. 2013. Measuring large-scale distributed systems: case of BitTorrent Mainline DHT. In *IEEE P2P 2013 Proceedings.* 1–10.

[19] Christopher Lemmer Webber, Jessica Tallon, Erin Shepherd, Amy Guy, and Evan Prodromou. 2018. ActivityPub. https://www.w3.org/TR/activitypub/

[20] Quanqing Xu, Zhiwen Song, Rick Siow Mong Goh, and Yongjun Li. 2018. Building an Ethereum and IPFS-Based Decentralized Social Network System. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS).* IEEE, 1–6. https://doi.org/10.1109/padsw.2018.8645058

[21] Hao Zhang, Yonggang Wen, Haiyong Xie, and Nenghai Yu. 2013. *Distributed Hash Table: Theory, Platforms and Applications.* Springer-Verlag GmbH. https://doi.org/10.1007/978-1-4614-9008-1