University of Reading

School of Mathematical, Physical and Computational Sciences

CS3IP16

Securing of Social Content with Non-Disruptive Cryptography

Student Name: Lucille Blumire

Student Number: 25010846

Supervisor: Hong Wei

Submission Date: July 25, 2019

ABSTRACT

This report describes and mathematically formalises a system which can be built on top of the fundamental axioms of public key cryptography as described by Rivest, Shamir, and Adleman. The system designed is intended to enable a robust system through which users might use public key encryption to encrypt content for multiple readers, with a focus on the design space of social media. This is achieved through the use of a zero-client-state authentication system, a trustful multiple reader encryption system (intended to aid in eliminated unethical usage), and a user-friendly public key–username substitution system.

ACKNOWLEDGEMENTS

I would like to thank Sascha M. Baer for his incredible assistance ensuring that all mathematical equations and descriptions were presented with a greater degree of clarity and succinctness.

I would like to thank John R. Spurr for his recommendation of using the dot operator as a cryptographic applicator in place of parenthesisation, which aids in clarity of understanding of cryptographic keys as more than just functions such that when they are themselves encrypted in the later parts of the mathematical description it does not feel unnatural.

I would like to thank the following people for proof-reading and providing editorial suggestions: David Avellan-Hultman, Ginevra Bonnaud, Santo Colli, Elysia Jewett, Beth Seymour, Helen Tyzack, Phillip Vanstone, and 1 other person who declined direct acknowledgement.

Finally, I would like to thank my family and friends for supporting me.

Contents

Glossary of Terms and Abbreviations

SocLocker

SocLocker, short for Social Locker, is the reference implementation of a social media solution powered by the cryptographic descriptions and theories outlined in this document.

Public Key Cryptosystem

A public key cryptosystem is cryptosystem that provides an asymmetric cryptography that uses two mathematical related keys. One of these keys is used to encrypt data and is called a public key, the other is used for decryption and is called a private key or secret key. Public keys are distributed publicly to allow any to use them to create a message readable only to the holder of the mathematically related secret key, which is kept secret—ensuring the security of the cryptosystem.

Mathematical Notations

Sections of this report rely heavily on mathematical notation. The following are less common or trivial notations which are used.

$$\forall x \in y.P$$

Means that for every element of x within a given set y a predicate P is true. Often P will be a predicate in terms of x.

$$A_b$$

Means an element of a set A identified by some key b. This might be numeric in the significance of an ordered set, or some unique identifier when referring to some form of key-value map.

$$F_p(n)$$

Means some function F which is defined with respect to a parameter p, is applied to a parameter n. This is often used in substitution for notation that would otherwise heavily rely on currying, i.e. $(F(p))(n)$ where F is a function applied to p, that is equal to a function that is applied to n. This subscript notation is used to improve readability.

$$A \cdot b$$

Means that a cryptographic key A is applied through sensible means to some message b. Usually encryption for a public key, and decryption for a private key, though often these operations are a symmetric mathematical application.

$$X\downarrow$$

Means that X is defined. Used to establish a predicate truth about the value of a function or membership of a set.

$$X\uparrow$$

Means that X is undefined. Used to establish a predicate truth about the value of a function or membership of a set.

1 Introduction

Social media stands at an intersection between the wild west of the internet of old, and the hyper-regulated internet some foresee for the future. It is only recently that both its content and the services themselves have begun being legislated and regulated. from persecution over online and digital crime, to requirements regarding transparency in usage of user tracking for advertising purposes.

In more recent years, it has become progressively more and more common that user privacy be sacrificed and the liberties and freedoms of users restricted. Not just from a legal perspective, but from content filtering and requirement to financially support content to reach a wider audience. No longer is social media equal, and no large western social media service provides a direct fashion for the viewing unfiltered of all content in chronological order without advertisements or native advertisements *without* jumping through some sort of hoop. This might be navigating to a page a user previously would not have to visit, or actively searching through settings and menus to configure one's own experience. That is, if it is even possible at all.

And though the service providers of social media are facing heavier and heavier legislation, most prominently in European law with laws such as the General Data Protection Regulation or GDPR coming into effect and changing the face of the internet substantially; social media mostly through deception, intentionally malicious user experience, or requirement for use, finds ways to subvert this and support an extremely profitable but privacy infringing modus operandi.

The users are not users. They are the product.

SocLocker—short for Social Locker—is an alternative social media solution which is designed and presented in this document. It runs on top of a protocol outlined herein and provides a user friendly, non disruptive experience, granting users total freedom, total control, and total liberation from the mishandling of their data. This is achieved through the usage of modern cryptographic techniques in tandem with the best practices of online security. The frameworks provided are generic, and SocLocker is just one potential reference implementation of the technology described in this document.

2 Problem Articulation & Technical Specification

The problem being solved is one of privacy and online security. Users of online social media platforms find their data being more and more often sold to advertisement companies, and their individual posts analysed and leaked to those they would rather not hold it. Security has proven to be a problem, with password leaks and security vulnerabilities happening across social media and effecting millions of users [1].

A specification for resolving this problem can be stated in a broader form, and then in a more targeted form specifically targetting the solution space that is explored in this report.

1.1 A users content should be accessible to only those whom they permit.

1.2 No agent other than the user themselves should be able to access or act on a users account and its associated data and content, without their express permission. Even in the event of a security breach.

1.3 A user should be able to create content, edit content, and access all content available to them, trivially without knowledge of the underlying system.

Each of these points can be addressed, and formulated into a more attainable and specific problem solution. Providing solutions from the perspective of limiting knowledge on a server level.

2.1 When a user creates content, they can specify exactly who should be permitted to access it. Then, only those people can access it. (follows from previous specification item 1.1)

2.2 The server should never handle user passwords. (follows from previous specification item 1.2)

2.3 The server should never handle non-encrypted user data. (follows from previous specification item 1.2)

2.4 The client should automatically handle encryption and decryption, with no knowledge of the process required by the server. (follows from previous specification item 1.3)

2.5 The client should not require express knowledge of other users cryptographic keys. (follows from previous specification item 1.3)

These can then be broken down once more, to give the overall feature spec presented by the technical specification.

3.1 A user is uniquely identified by a secret key that at all times only they (any anyone they give this key to, who can then act as them) know.

3.2 Actions taken on behalf of the user require proof of knowledge of their secret key.

3.3 The server will never require direct knowledge of a secret key.

3.4 A user is able to create content and submit it to be accessed by a finite set of users.

3.5 The server, and users who are not permitted on a content by content basis, should not be able to access any created content.

3.6 A user should be able to make corrections and alter their previously created content.

3 Literature Review

A number of end-to-end encrypted social networks have come and gone. Many end up never being fully completed, and others end up dying soon after launch. This is because of the one core problem with removing the advertisement and data-selling components of social media.

They have no alternative profit stream.

As a result, there is no large plethora of content to review, and it is difficult to find full papers or even blog posts outlining the cryptographic techniques and methods used by other similar projects.

As such, this literature review will serve as a much less in depth, but hopefully broader overview of the existing technology and the academic papers and articles available relating to it.

3.1 RSA

RSA [2] provides an outline of the skeleton of cryptography. Being one of the first public-key cryptosystems—published in 1977—to be proposed and documented, it's provided a core basis to the field that most other works build on top of.

The RSA cryptosystem mandates the creation of a public key which is based on two large prime numbers. They then publish their own public key, and keep the two large prime numbers private. Content can be encrypted with their public key, which can then only be decrypted by a someone who knows which two large prime numbers were used to create it. This means that breaking through RSA encryption is related to the factoring problem, but whether it is directly as difficult is an open problem in computer science.

One obvious downside of the system, is that for the substantially large keys that are required for encryption and decryption, the entire process is relatively slow.

The initial model of public-key cryptography which is presented by RSA, including most of its axioms, is definitely something that could be extended upon for the purpose of the project. Most further developments in cryptography build on top of the axioms presented by RSA, and thus by having a system designed with these axioms in mind the encryption systems that are used in any given implementation can be kept abstract and changed to suit the needs of any given system.

3.2 Keybase

Keybase is primarily an identity verification service—it allows a user to prove through cryptographic methods that their facebook account, twitter account, any content verified by a given PGP key, and many other social platforms, are all owned by the same person. As such it is not directly in the same space as this project, but might contain aspects that can be learned from and evaluated. It does however have an integrated messaging and group messaging service, which is much more in the space of this project.

It's usage of cryptography [3] in its messaging service relies on NaCl [4] for its encryption, a cryptographic system that will be explored further later in this literature review in Section 3.3.

For it's identity and key ownership verification, Keybase takes an appropriately paranoid approach: The server trusts nothing, the client trusts nothing, everything is independently verified.

This protects Keybase from a number of higher level security concerns, such as the server being compromised and transmitting malicious data such as corrupted client-side code and deliberately misleading identity data.

In order to protect against some of this, certain rules are in place about how user identities are managed. Primarily they are stored in a chain that grows monotonically and is never rolled back. This has certain implications with regards to the European Unions Right to be Forgotten however, which will be discussed in Section 10. The guarantee it does provide is that once the server signs and verifies a change to a user identity chain (the historical account of all their identities, revoked or otherwise), this is known and persistently viable to any other user.

Another technique used by Keybase is to directly store their Merkle Root (essentially, a hash) to the Bitcoin block-chain. As Bitcoin is well protected against forking through its proof-of-work verification strategy [5], that makes falsifying a fork of Keybase remarkably difficult as it would require compromising one of the largest public blockchain networks and cryptocurrencies.

The downside of using blockchain technologies such as bitcoin is that they require funding in order to interface with, which is something that ideally should be minimised for the maintenance of this project: in an ideal world, the only express costs would be server upkeep.

Much of the complexity required for Keybase draws from its entirely public facing model, in which outside of its message system, the functions it performs for identity verification are public and can be depended upon. Thus the complexity of this system can be reduced and many of the security considerations of Keybase can be made unnecessary by considering that this projects goal is to provide complete privacy of content sent, not complete verification of identity of sender or receiver. Therefore this system could be made intentionally trustful.

3.3 NaCl

NaCl is a powerful networking and cryptographic library [4] that is used in Keybase (as discussed in Section 3.2) for its messaging service.

This provides a tool which is called `crypto_box` which allows for public key authenticated encryption. It signs all messages encrypted, and thus for any given message the intended receiver is the only user who can access the content, and they can verify that it is sent by the sender who claims to have sent it.

This tool is built on top of Salsa20[6], Poly1305[7], and curve25519[8]. These are technologies that have been worked on by Daniel J. Bernstein, who is the developer of NaCl.

The overall intention behind the design of NaCl is to provide a high level, easily usable abstraction to common cryptographic primitives. In this respect, it aims to provide security by default, and enable easy development of encrypted systems without the requirement of complex configuration and therefore it minimises the room for misconfiguration and vulnerabilities.

Further to this, it's implementation as a public key cryptosystem fulfils the first three axioms defined in the original RSA paper, making it a highly compatible drop in replacement for many existing systems and making it easy to test generic cryptographic designs in a way that has a high degree of compatibility.

It has a highly secure [9] basis and is cryptographically robust. In addition to this, it achieves that robustness without extensively sacrificing speed of encryption, making it ideal for real time applications such as social media with a focus on non disruptive cryptography.

4 Solution Approach

There are a number of technical issues which result from the technical specification. The approach taken to solving these is one with a strong mathematical basis. By building a rigorous definition of some mathematical functions which uphold given invariants.

A function κ exists which encrypts a message μ for a set of users X. Another function δ exists which decrypts a message encrypted by κ if and only if it is also provided with a valid public and private key pair for a user specified in X.

In addition to this, a method must exist to validate that a user possesses a given private key without them directly supplying such a private key.

Specification 3.1 is fully resolved by the validation method described above, as it eliminates the need for server knowledge of such a key.

Specification 3.2 is fully resolved by requiring that for any server modifying action taken by a user they are required to fulfil the validation method described above.

Specification 3.3 is fully resolved by the given requirement for the validation method that it does not require a private key to be directly supplied, and that κ can be evaluated at least partially on a client machine to an extent where their private key is unrecoverable when the remainder of the function is evaluated on the server.

Specification 3.4 is fully resolved by the given definition of κ and the ability to submit such fully encrypted content in a retrievable format to the server.

Specification 3.5 is fully resolved by the given definition provided the if and only if clause of δ can be satisfied.

Specification 3.6 is fully resolved provided that a user can re-evaluate κ for an unchanged X and have the server acknowledge the change made in a fashion that destroys the original content.

Thus, in order to achieve the above results, a mathematical expression for κ and δ will be demonstrated in Section 5. This is not implemented from a purely mathematical perspective, as it will require mutation of a persistent state. This is however performed on the server, and will not invalidate any of the specification fulfilment's set out in this section.

From an architectural standpoint, security through public access will be used. This is a philosophy in which all clients have near full access to the servers content. This requires security to be handled external to the server, and reduces the risk if the server or an administrator of it are directly compromised. Direct database writing will still provide a potential security risk, but as no non-encrypted content is expected to be stored it should be impossible for significant falsifications to be performed even by a rogue administrator.

5 Implementation: Mathematical Description

5.1 Public Key Cryptosystems

Let us consider a public key cryptosystem that is defined such that each user $u \in U$ where U is the set of all users, has a pair of cryptographic keys $(C_u, \bar{C}_u)$. C_u represents a cryptographic public key, $\bar{C}_u$ represents a cryptographic private key.

These are applied to data, encrypting it. This application of a cryptographic key on the left hand side to data on the right is represented by a dot operator $\cdot$.

μ is used to represent any arbitrary message that might be encrypted or decrypted. It should be noted that it is entirely possible for this to be a cryptographic key.

$$\forall u \in U.\ \bar{C}_u \cdot \mu \neq \mu \tag{1}$$

This notation is right associative such that

$$\forall \alpha \in U.\ \forall \beta \in U.\ C_\alpha \cdot C_\beta \cdot \mu \equiv C_\alpha \cdot (C_\beta \cdot \mu) \tag{2}$$

Next, let us define a set of fundamental rules for the operation of our public key cryptosystem. These axioms were originally defined in 'A method for obtaining digital signatures and public-key cryptosystems' by R. L. Rivest, A. Shamir, and L. Aldeman in their famous discovery of the RSA cryptosystem[2]. It should be noted that our system does not require axiom (d) from their paper, and thus the implementation cryptosystem chosen does not meet it.

(a) Deciphering the enciphered form of a message μ yields μ. Formally,

$$\forall u \in U.\ \bar{C}_u \cdot C_u \cdot \mu = \mu \tag{3}$$

(b) Both C_u and $\bar{C}_u$ are easy to compute.

(c) By publicly revealing C_u, the user $u \in U$ does not reveal an easy way to compute $\bar{C}_u$. This means that in practice only they can decrypt messages encrypted with C_u, or compute $\bar{C}_u$ efficiently.

5.2 Multiple Key Composition

In order to make a message readable by multiple people, it is necessary to compose multiple instances of the message with multiple keys.

The objective of this is to allow anyone with an appropriate key to read a message.

For a non-empty finite unordered set of input cryptographic functions X, let κ_X be a function which composes the cryptographic functions in X and $\delta_{C_\alpha,\bar{C}_\alpha}$ be a function which decrypts that composition for a user $\alpha \in U$ if and only if their public key C_α is a member of the set X of permitted decryptees.

$$\forall u \in U.\ \delta_{C_u,\bar{C}_u}(\kappa_X(\mu)) = \mu \iff (C_u \in X) \tag{4}$$

A naive approach to implementing δ and κ as defined above would be to encrypt the message μ with each element of X and distribute it individually to each user. This creates an issue however, for if the data is to be edited or the users who are to access it are modified, then the data must be re-encrypted and sent to every member of X.

An alternative solution is to use a forwarding technique, whereby each message μ is encrypted with cryptographic functions generated at the point of creation $(C_\pi, \bar{C}_\pi)$. The secret key is then encrypted $C_{\text{recipient}} \cdot \bar{C}_\pi$ and sent to each recipient, in order to access the data. This way, any modifications to the data does not require a fully re-encrypting it for each recipient. Each C_π is likely to be unique for any given encrypted message μ.

To implement this, we must consider the set Σ which might have determined value for any post μ and a public key C_u where $u \in U$. The value will be the encrypted cryptographic function $C_u \cdot \bar{C}_\pi$ which when decrypted by the secret key $\bar{C}_u$ corresponding to the public key it is indexed by will produce the secret key $\bar{C}_\pi$ corresponding to the public key the message μ was encrypted with. In instances where a user with public key C_u has permission to access a message μ, this value will be defined. In all other instances it is undefined. The cryptographic function given by decrypting an element of Σ is always the cryptographic inverse of the cryptographic function its second element was encrypted by.

Note that $\downarrow$ is read 'is defined' and $\uparrow$ is read 'is not defined'.

$$\forall u \in U.\ \Sigma_{C_u, C_\pi \cdot \mu}\downarrow \implies \Sigma_{C_u, C_\pi \cdot \mu} = C_u \cdot \bar{C}_\pi \tag{5}$$

With this given, we can assert the following rule.

$$\begin{aligned} &\forall u \in U. \\ (\bar{C}_u \cdot \Sigma_{C_u, C_\pi \cdot \mu}) \cdot C_\pi \cdot \mu &= (\bar{C}_u \cdot C_u \cdot \bar{C}_\pi) \cdot C_\pi \cdot \mu \\ &= \bar{C}_\pi \cdot C_\pi \cdot \mu \\ &= \mu \iff \Sigma_{C_u, C_\pi \cdot \mu}\downarrow \end{aligned} \tag{6}$$

In order for this system to function, we must allow mutation of the set Σ to contain new elements as and when a user is granted access to a message.

This is done *implicitly* at the point of message encryption for a set of users, which is done by the κ_X function where X is a set of cryptographic public keys that should be granted permission to read the message encrypted by κ_X.

We can give the following definition for the value of $\kappa_X(\mu)$.

$$\kappa_X(\mu) = C_\pi \cdot \mu \tag{7}$$

And the following explanation of the mutation on Σ performed for each instance of κ_X that occurs.

$$\begin{aligned}
&\text{if } \kappa_X(\mu) \text{ is used} \\
&\qquad \forall \xi \in X. \\
&\qquad \Sigma_{\xi,\kappa_X(M)} := \xi \cdot \bar{C}_\pi \\
&\therefore \forall u \in U.\ C_u \in X \equiv \Sigma_{C_u,\kappa_X(M)} \downarrow
\end{aligned} \tag{8}$$

This allows us to write a definition of δ that fulfils the iff predicate of equation 4. Where C_α and $\bar{C}_\beta$ are cryptographic functions and ϵ is an encrypted message.

$$\forall \alpha \in U.\ \forall \beta \in U.\ \delta_{C_\alpha,\bar{C}_\beta}(\epsilon) = (\bar{C}_\beta \cdot \Sigma_{C_\alpha,\epsilon}) \cdot \epsilon \tag{9}$$

The following combined all of the above definitions to demonstrate that the definitions of κ and δ satisfy the required constraints.

$$\begin{aligned}
&\forall u \in U. \\
\delta_{C_u,\bar{C}_u}(\kappa_X(\mu)) &= \delta_{C_u,\bar{C}_u}(C_\pi \cdot \mu) \\
&= (\bar{C}_u \cdot \Sigma_{C_u,C_\pi \cdot \mu}) \cdot C_\pi \cdot \mu \\
&= (\bar{C}_u \cdot C_u \cdot \bar{C}_\pi) \cdot C_\pi \cdot \mu \\
&= \bar{C}_\pi \cdot C_\pi \cdot \mu \\
&= \mu \iff (C_u \in X \equiv \Sigma_{C_u,C_\pi(\mu)} \downarrow)
\end{aligned} \tag{10}$$

5.3 Implementation Mathematical Variation

The reference implementation will differ from the pure notions of a public key cryptosystem described above on a number of accounts. This is due to the chosen backing cryptosystem, and modifications would need to be made to the mathematical descriptors above for any such system. However, the modifications made are solely to the information required for a cryptographic application. Given that the information needed for those parameters is trivially available as described in 7.1, the reference implementation will fulfil the constraints of depending fundamentally only on the axioms and definitions above.

The implementation used is a combination of Salsa20[6], Poly1305[7], and curve25519[8]. In combination, these are used as part of the NaCl cryptography library[4] which is discussed further in Section 3.3.

This specific cryptosystem was chosen due to its ease of use and underlying security[9]. In addition to this, it meets all the mathematical requirements set forward by the axioms defined in 5.1.

The new information required for cryptographic application is defined beneath, with $S \in U \wedge R \in U$ being the sender S and intended recipient R of an encrypted message.

1) All encryption must be signed, requiring knowledge of a senders own $\bar{C}_S$ and the recipients C_R for all encryption, and the knowledge of the senders C_E for decryption.
2) All encryption requires a nonce, decryption requires the nonce used for encryption.

Formally where μ is the message to encrypt, ν is a randomly generated nonce, C_R is the public key of the recipient, and C_S is the private key of the sender, box is the NaCl encryption method, and open is the NaCl decryption method.

$$\begin{aligned} &\forall R \in U. \\ &\forall S \in U. \\ \bar{C}_R \cdot C_R \cdot \mu = \text{open}(&\text{box}(\mu, \nu, C_R, \bar{C}_S), \nu, C_S, \bar{C}_R) \end{aligned} \tag{11}$$

6 Implementation: Server Architecture

6.1 Architecture

The server will be implemented using Rocket https://rocket.rs, a web application programming library for the Rust programming language https://rust-lang.org. This is due to its endpoint driven functional approach that will enable quick and easy implementation of all core functionality. Rust is selected due to its secure implementations of the NaCl cryptographic libraries https://github.com/sodiumoxide/sodiumoxide, and ease of writing correct code when contrasted with similar languages such as C and C++.

6.2 Database Architecture

The database is designed around four tables which provide all the defined functionality of the service. These are `Users` which provides a mapping from `Username`s to `PublicKey`s; `Posts` which store the encrypted content of each post; `NOA` which stores the notices of access (Σ) for each user to each post, and their encrypted access keys; and `Auth` which is used to provide authentication tokens.

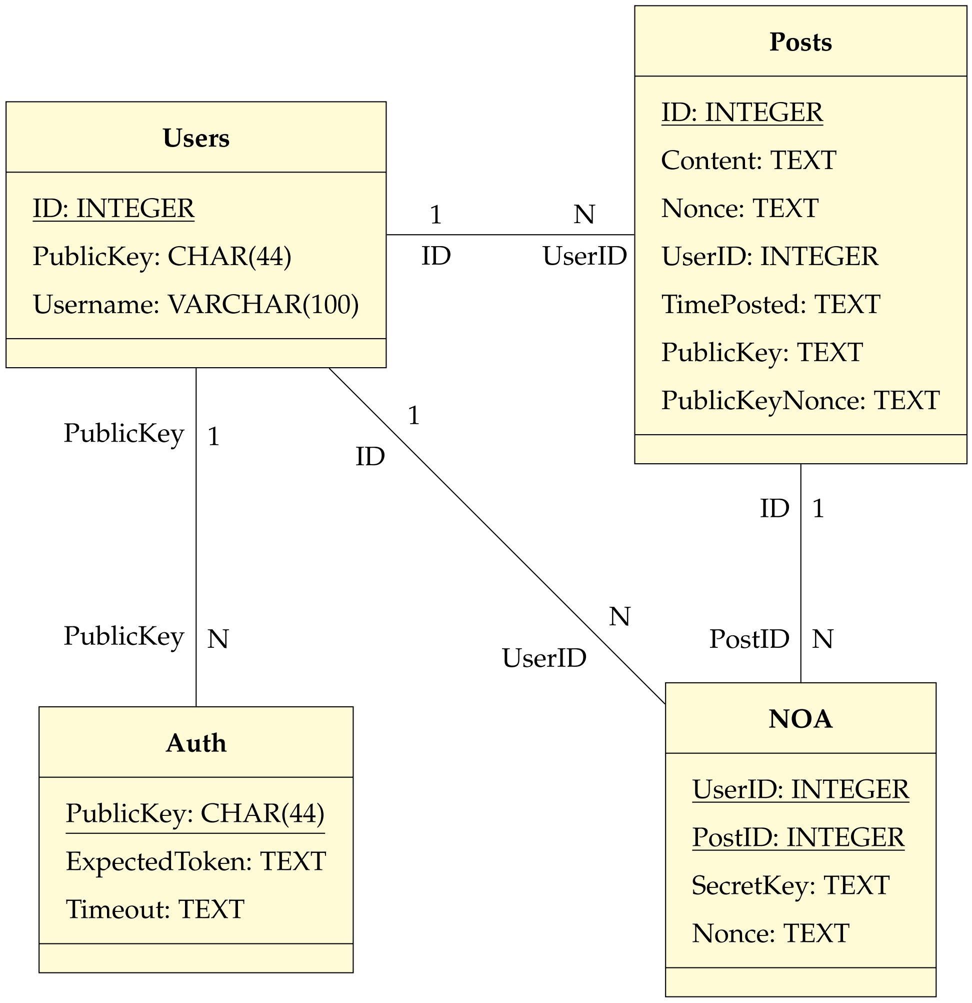


Fig. 1. Database Relationship Diagram

6.3 User Account Creation

A user account is simply a mapping between a username and a public key, that other users can look up. As such the process of creation is a trivial database insertion operation. If the given username already exists then the operation fails.

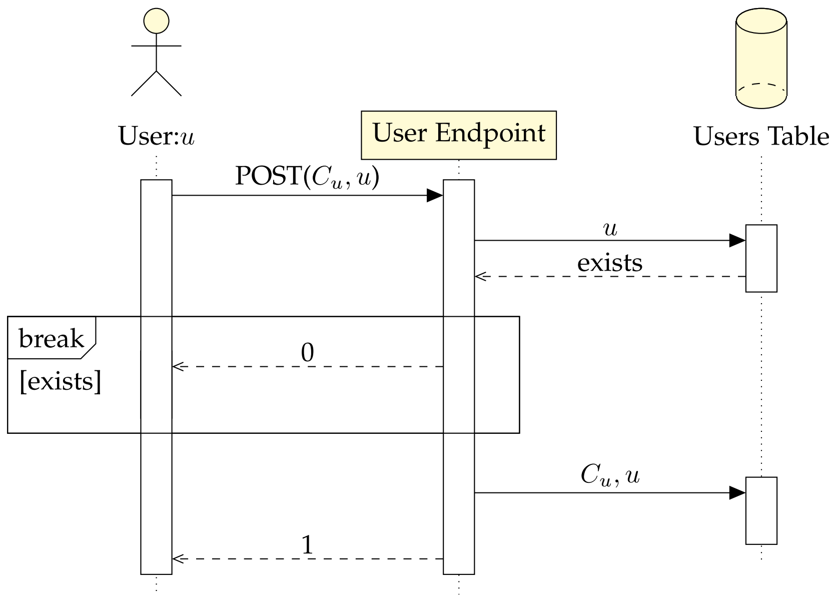


Fig. 2. User Registration

This endpoint will return

409 if the username already exists.

201 if account creation is successful.

500 if for whatever reason the database operations fail.

A successful request would have the following format.

```
{
    "publicKey": "... (base 64 encoded)",
    "username": "..."
}
```

An endpoint is then exposed to allow accessing of a users public key from their username.

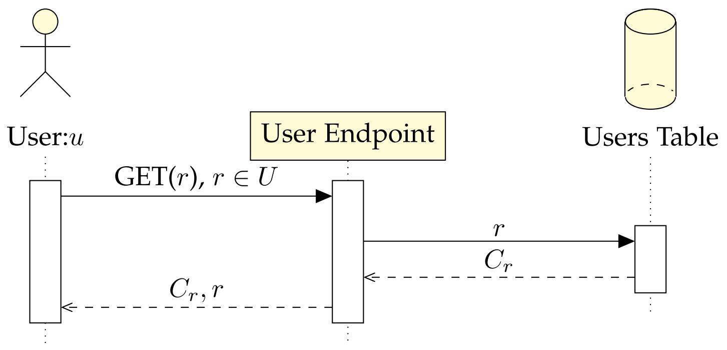


Fig. 3. User Public Key Query

The endpoint will return

200 with a body if the username exists.

404 if the username does not exist.

A successful request would simple use the query parameter of the url, requesting the endpoint `/user?username=...`. The body on a successful response will be.

```
{
    "id": 0,
    "publicKey": "... (base 64 encoded)",
    "username": "..."
}
```

6.4 Authentication Layer

Authentication remains an important part of the system, although fully impersonating a user without their secret key is impossible, it is desirable to avoid garbage encrypted attempts at impersonation from filling the database. This is done via providing an authentication endpoint that is accessed and used for verification purposes, and sits like a layer that all other transactions must pass through.

It is also possible to perform a trivial authentication (with no other operation) in order for a client to verify a user login internally.

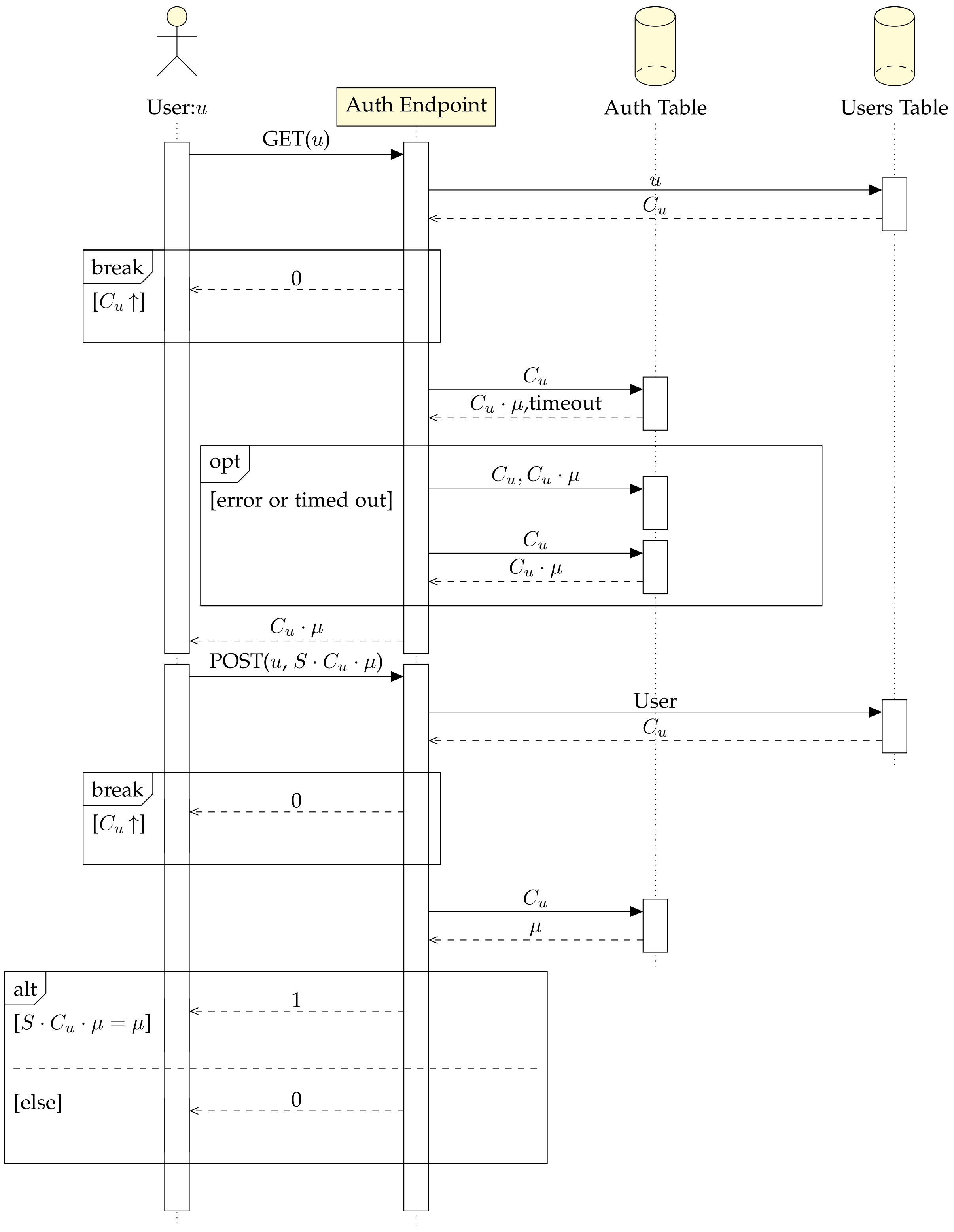


Fig. 4. Authentication Sequence Diagram

The endpoint on a get request will respond

200 with a body if the user exists.

404 if the user does not exist.

500 if there is an issue with database operations.

It expects a request with a query string, `/auth?username=...`. It will respond with the following body on success.

```
{
    "encryptedToken": "... (base 64 encoded)",
    "nonce": "... (base 64 encoded)"
}
```

On a post to the endpoint, the server will respond `200 OK` with a body of either true if the validation is successful, or false if the validation is not. A successful post will have the following format.

```
{
    "decryptedToken": "... (base 64 encoded)",
    "username": "..."
}
```

6.5 Accessing Posts

In order to view social content a user must have a way of discovering what content is accessible to them. This is done through accessing the `noa` endpoint.

This follows a much more complicated structure, but its overarching sequence diagram is as follows.

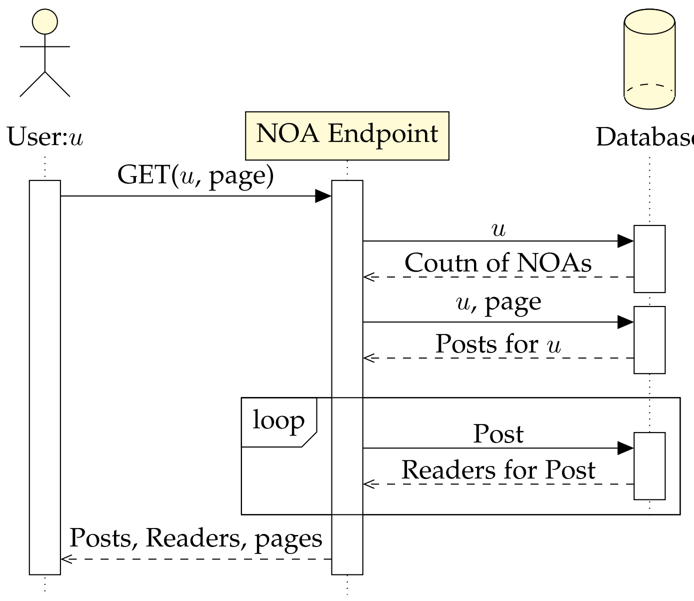


Fig. 5. Notice of Access Endpoint

This is summarised above as the query performed is substantially more complicated than in previous sequences.

```
SELECT
    NOA.SecretKey as NOAEncryptedSecretKey,
    NOA.Nonce as NOAEncryptedSecretKeyNonce,
    Posts.Content as PostEncryptedContent,
    Posts.Nonce as PostNonce,
    Posts.ID as PostID,
    Posts.TimePosted as PostTimePosted,
    Posts.PublicKey as PostEncryptedPublicKey,
    Posts.PublicKeyNonce as PostEncryptedPublicKeyNonce,
    Author.Username as AuthorUsername,
    Author.PublicKey as AuthorPublicKey
FROM
    NOA,
    Users as Recipient,
    Users as Author,
    Posts
WHERE
    NOA.UserID = Recipient.ID AND
    NOA.PostID = Posts.ID AND
    Posts.UserID = Author.ID AND
    Recipient.Username = {username}
ORDER BY
    Posts.TimePosted
LIMIT
    25
OFFSET
    {page * 25};
```

A subsequent query is then run once for each row, to acquire each of the other users who has shared access to each post.

```
SELECT DISTINCT
```

```
    Recipient.Username
FROM
    NOA,
    Users as Recipient,
    Posts
WHERE
    NOA.UserID = Recipient.ID AND
    NOA.PostID = {post_id};
```

The number of pages is acquired through the following query.

```
SELECT
    COUNT(NOA.PostID)
FROM
    NOA,
    Users as Recipient
WHERE
    NOA.UserID = Recipient.ID,
    Recipient.Username = {username};
```

The NOA endpoint will respond

200 with a body. In the event the user does not exist, this simply returns a body with no notices of access.

The endpoint should be queries with `/noa?username=...&skip=0` incrementing skip to move to higher pages. It will resopnd with the following, with the `noas` record containing as many elements as are in the list, one is given as an example. If the user does not exist, has no posts to read or there is an error, the list will simply be empty. `allReaders` will also be a list containing likely more than one element.

```
{
   "noas": [
      {
         "post": {
            "encryptedContent": "... (base 64 encoded)",
            "nonce": "... (base 64 encoded)",
            "username": "...",
                "publicKey": "... (base 64 encoded)",
                "postId": 0,
                "timePosted": "YYYY-MM-DDThh:mm:ss",
                 ↪  "encryptedPublicKey": "... (base 64 encoded)",
                "encryptedPublicKeyNonce": "... (base 64 encoded)"
            },
            "encryptedSecretKey": "... (base 64 encoded)",
            "nonce": "... (base 64 encoded)",
            "allReaders": [
                "...",
            ]
        }
    ],
    "pages": 0
}
```

6.6 Creating a Post

When creating a post, the user supplies the encrypted content of the post, the nonce, and encrypted key and username for each user they wish to provide a notice of access. The user must also supply an authentication proof that is then verified using the authentication layer.

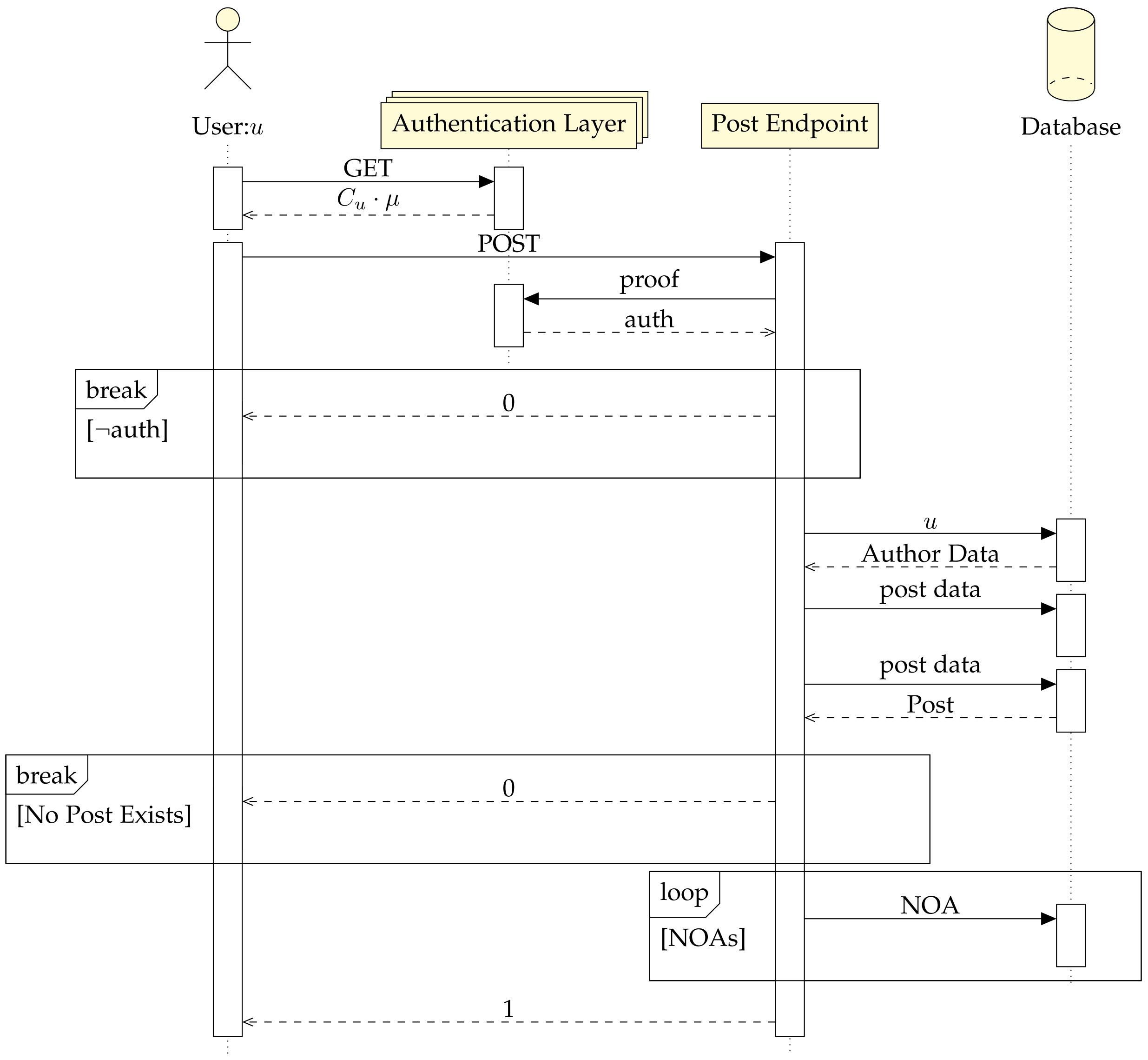


Fig. 6. Creating a Post

Creating a post involves posting to the post endpoint. It will return

200 with a body of true if the post is created successfully.

403 if the user does not provide a correct proof.

500 if there are database operation errors.

A successful post might look like this, with `noaEncryptedKeys` containing a record for each user being authorised to read the post.

```
{
    "content": "... (base 64 encoded)",
    "noaEncryptedKeys": [
        {
                "encryptedSecretKey": "... (base 64 encoded)",
                "nonce": "... (base 64 encoded)",
                "username": "...",
            }
        ],
        "nonce": "... (base 64 encoded)",
        "proof": "... (base 64 encoded)",
        "publicKey": "... (base 64 encoded)",
        "publicKeyNonce": "... (base 64 encoded)",
        "username": "..."
}
```

6.7 Altering a Post

Altering a post is much simpler than creating a new one, and requires only the posts ID, the new encrypted content, the new nonce, and a proof of authentication.

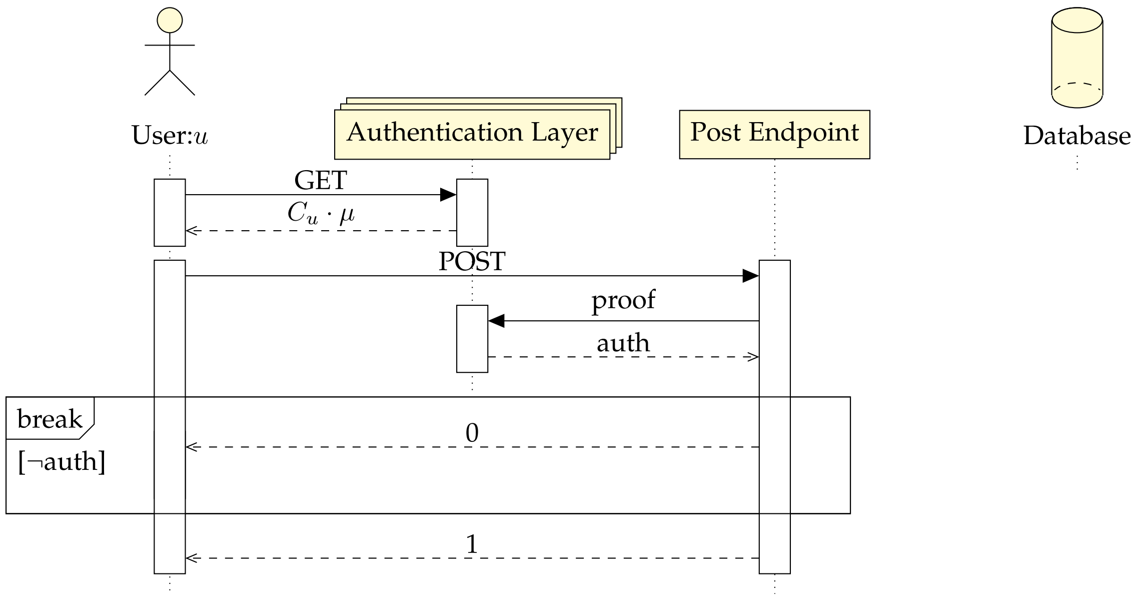


Fig. 7. Altering a Post

The endpoint behaves similarly to the creation interface, returning

200 on success.

203 on invalid authentication.

404 if the post does not exist.

500 if there is a database error.

A successful put request body for the post endpoint would resemble this:

```
{
    "newContent": "... (base 64 encoded)",
    "newNonce": "... (base 64 encoded)",
    "postId": 0,
    "proof": "... (base 64 encoded)"
}
```

6.8 Server Public Key

For reasons described in section 7.1, a means of accessing the servers public key must be provided.

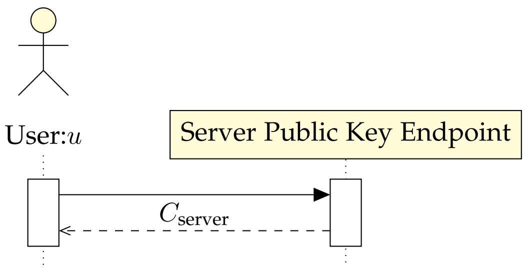


Fig. 8. Accessing the Server Public Key

The endpoint expects a get request, and responds with `200 OK` and a body of the base 64 encoded server public key.

7 Implementation: Client Processes

7.1 Knowledge Acquisition

There are a number of differences between the NaCl cryptosystem used and the axioms defined by RSA, as outlined in 5.3. These differences are resolved through suitible knowledge acquisition, which can be performed by the client. Each required piece of information is obtained as follows.

It is assumed a user of the system knows the following:

- Their own public key
- Their own private key
- A set of usernames of themselves and other users

In order to use the system, they require the following additional information

- A set of Public Keys of other users
- The servers own public key

7.1.1 Arbitrary Public Keys

In order to make content available to someone, you require their public key, as well as your own private key. Users will know the username of the person they wish to make content available to, and thus the client can simply access the endpoint described in figure 3.

7.1.2 The Servers Own Public Key

In order to perform authentication, the user must know the public key of the server which is used to verify the signatures on the encrypted authentication proofs, which are needed for the client to properly interface with the Authentication Layer described in figure 4. Access to the servers public key is described in figure 8.

7.2 Architecture

The client interface is built using modern web technologies, such as more recent versions of ECMAScript and CSS. This is translated for compatibility through Babel https://babeljs.io/, and is built with TypeScript https://www.typescriptlang.org/ on top of the Vue JS https://vuejs.org/ framework. The interface was built using Vuetify https://vuetifyjs.com/, which together with Vue provides a versatile and modular component driven architecture for assembling a web application, that meets the modern standards of material design.

7.3 Plugins

A number of additional plugins were used to bring the app together. Each of these provides additional functionality that is relied upon by the rest of the system.

7.3.1 Auther

Auther was a component I wrote and included in the design of the application to abstract and represent interfacing with the authentication layer defined in section 6.4. It is called from many places and serves to get and decode a proof from the server, handling the users side of the authentication layer.

7.3.2 Router

There are two core routes which are used by SocLocker. The first is the public route, which displayed information about the service being provided. This is located on `/` and is connected to the `PublicView` component described in section 7.4.1. The second is the feed route, which shows a logged in user the posts available to them and provides them a way to create new posts. This is located on `/feed` and is connected to the `Feed` component described in section 7.4.2.

7.3.3 Store

The store is a persistent client-side datastore that is used throughout the clients interaction with the service. Its primary function is to store the information of a logged in user, and to thus handle the process of acquiring and validating this information on login. Stored within it are the following:

- `username` — The logged in users username
- `publicKey` — The public key acquired from the `/user` endpoint for the users username.
- `secretKey` — The key that should be applied and used when accessing the `/auth` endpoint for verification.
- `userId` — The users unique identifier.
- `loginAttemptFails` — Used for tracking a users attempts at logging in, and as a simple messaging service to enable notification of successful login.
- `isLoggedIn` — A boolean state indicating the validity of the rest of the data. While false the rest of the data is assumed to be empty (empty strings, zero, empty arrays, etc).

The login process is also handled by the store, which contacts the `/user` and `/auth` endpoints described in sections 6.3 and 6.4 respectively to validate a users login as successful.

7.4 Views

Views provide the main means of displaying content to the user. They act like pages in the design of a Vue Js system. There is a single piece of functionality present on all views, and that is the `TopBar` component described in section 7.5.6, as well as the functionality to load the dialogs `Login`, `Register`, and `Logout`, which are described in 7.5.2, 7.5.5, and 7.5.3 respectively.

7.4.1 PublicView

The Public View is available to all users. Logged in or not. The content on this page is built entirely to render a simple timeline of events that occur when creating content on the website, and serves as the core form of marketing for the platform present on the website. It provides useful links to users wishing to understand the cryptographic procedures better.

7.4.2 Feed

The feed performs the bulk of the work that a user will interface with on the website. Consisting initially of a snack-bars (small single line popup) that can appear at the top of the screen notifying the user any errors when loading the posts. These might occur if for any reason the API server goes down but the public interface is still functioning.

It then contains the `CreatePost` component (section 7.5.1, to allow a user to create a new instance of a post. After this, follows a paginated interface that contains as many `PostBox` components (section 7.5.4) as are acquired for readable posts on each page.

On load, the page first checks if the user is logged in. If they are not they are immediately redirected to the `PublicView` (section 7.4.1 page. If they are not redirected and remain on the page, the process of accessing and displaying all of the posts accessible to a user is undergone. This is done through repeated (over a certain repetition delay) calls to the `/noa` endpoint described in section 6.5. Each post is then displayed inside its own `PostBox`. The skip component of the query to `/noa` is incremented based on the page being accessed through the pagination interface.

7.5 Components

Components are modular parts that are combined in order to build up a Vue JS website. In this specific design, and can be displayed across a variety of pages.

7.5.1 *CreatePost*

The `CreatePost` endpoint serves to provide the user an interface with which they are able to create, encrypt, and submit social content. It presents them with a text area into which they can type the post they are creating, as well as another area to type users they wish to authorise to access the post. This automatically contains their own username which cannot be removed. Though the server supports creating a post that the user themselves cannot access, this is likely undesired behaviour in a social media context and so it forbidden client side.

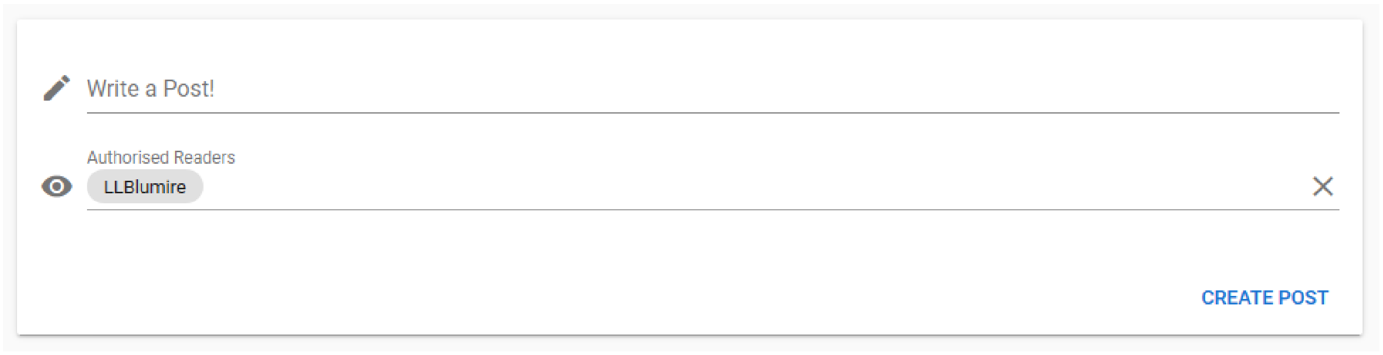


Fig. 9. CreatePost Component

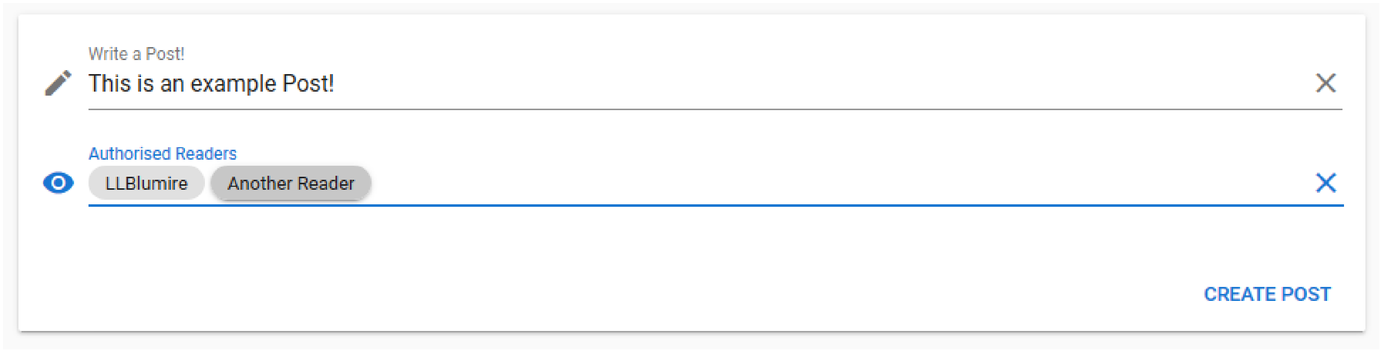


Fig. 10. CreatePost Component In Use

When the post is submitted by clicking the 'CREATE POST' button, the client generates a key, encrypts the content of the post with that key, accesses the public key for each provided username, and encrypts the decryption key for access by each user. This is all then submitted, alongside a proof of authentication, to the `post` interface as described in section 6.6. Any errors are reported in a snack-bar which is shown at the top of the screen.

7.5.2 Login

The login interface provides a large group of snack-bars which are used to notify the user of successful login, login failure, and invalid login form submission.

It then provides most of its functionality in a dialog, which contians a boxes for users to enter their username and password.

The functionality for this is the offloaded to the Store (section 7.3.3). The component monitors the `loginAttemptFails` value and displays notification of failure whenever it increments, and otherwise monitors for successful login (`isLoggedIn` changing from false to true).

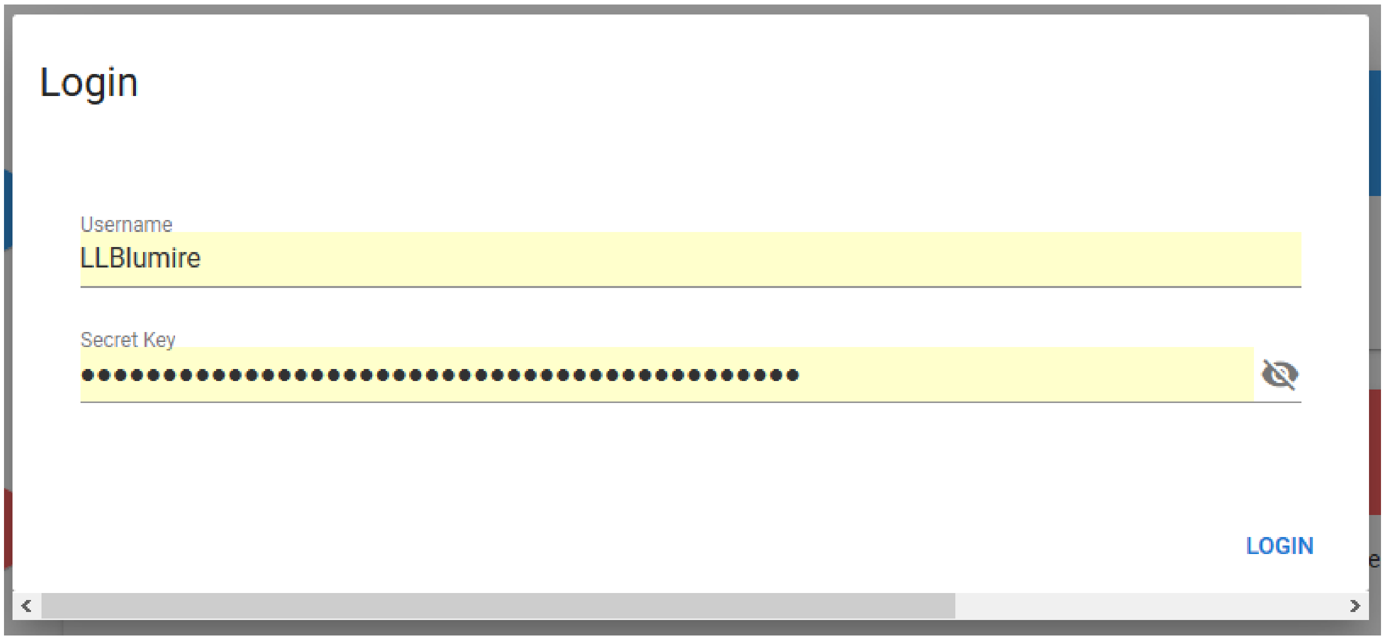


Fig. 11. Login Component In Use

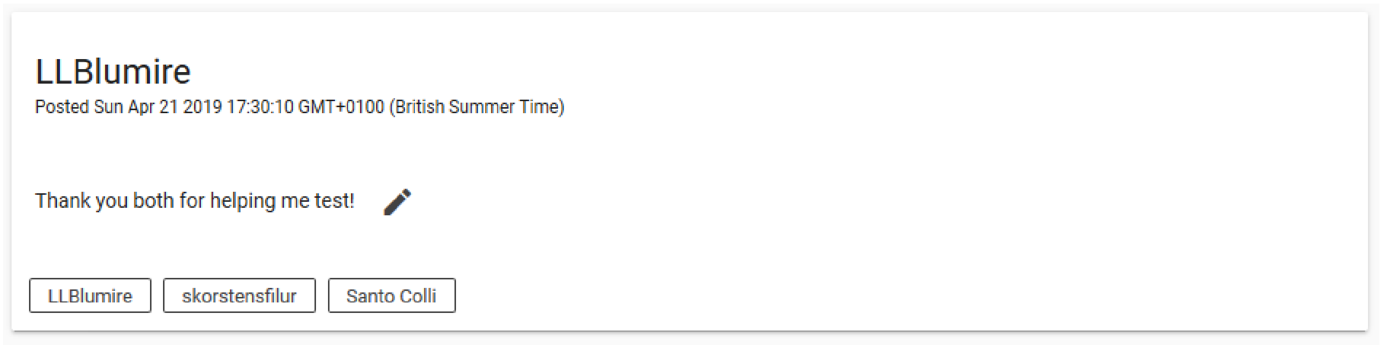
7.5.3 Logout

The `Logout` component is the most trivial within the system, and provides a single function which offloads logging out to the store (section 7.3.3) which simply performs a client side resetting of login data. It is implemented as a component rather than a single function exposed as a plugin like `Auther` simply for symmetry with `Login` and `Register` (sections 7.5.2 and 7.5.5 respectively).

7.5.4 PostBox

The `PostBox` component displays information about a post that a user has access to. They are also responsible for issuing alterations to a post by sending put requests to the `/post` endpoint as described in 6.7.

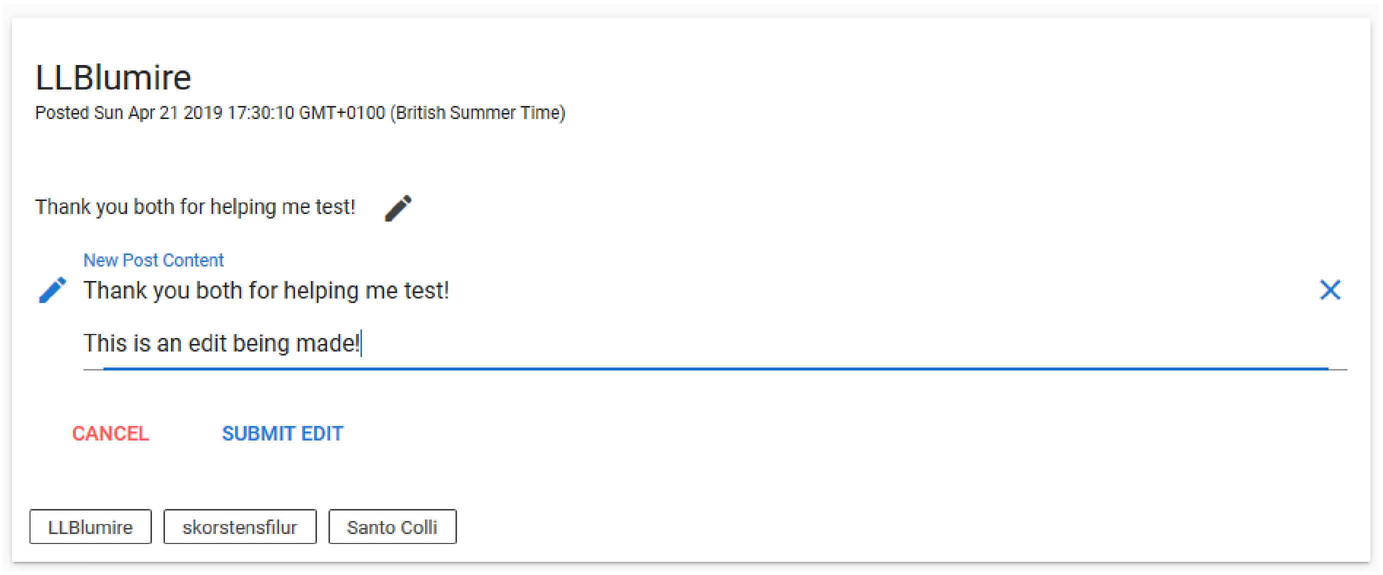
The post box displays the user who posted it, the date it was contributed (with timezone adjustments for the local reader), the content of their message, an option to edit the post, and a list of all the users who are able to access the post.



LLBlumire
Posted Sun Apr 21 2019 17:30:10 GMT+0100 (British Summer Time)

Thank you both for helping me test!

Fig. 12. PostBox Component



LLBlumire
Posted Sun Apr 21 2019 17:30:10 GMT+0100 (British Summer Time)

Thank you both for helping me test!

New Post Content
Thank you both for helping me test!

This is an edit being made!

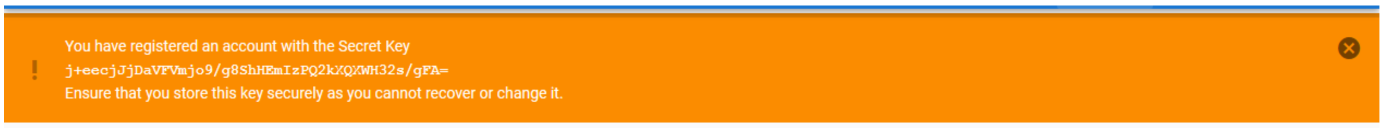
Fig. 13. PostBox Component Altering Post

7.5.5 Register

Registration is largely handled by posting to the `/user` as described in section 6.3.

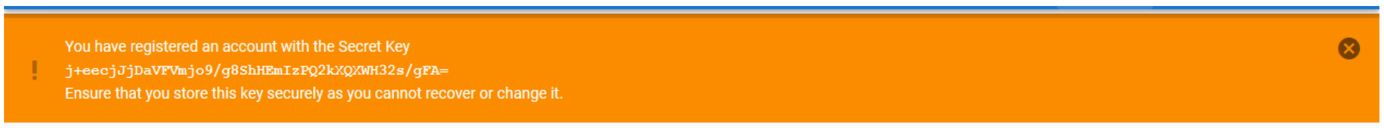
The `Register` component displays a dialog and simply generates keys as displays them to the user, giving them the option to generate a new pair until they are satisfied, at which point they can assign as username and register.

Snack-bars are used to inform the user of successful and unsuccessful registration. In addition to this, post registration an alert component is generated in order to give the user one final opportunity to access and store their private key.



You have registered an account with the Secret Key
`j+eecjJjDaVFVmjo9/g8ShHEmIzPQ2kXQXWH32s/gFA=`
Ensure that you store this key securely as you cannot recover or change it.

Fig. 14. Register Component



You have registered an account with the Secret Key
`j+eecjJjDaVFVmjo9/g8ShHEmIzPQ2kXQXWH32s/gFA=`
Ensure that you store this key securely as you cannot recover or change it.

Fig. 15. Register Component Alert Box

7.5.6 TopBar

The `TopBar` component serves as the central hub for both navigation and account operations. Providing the interface for opening the login and registration component dialogues, and triggering the logout component. It has two main modes, which alter what information is displayed. While logged out, a user can login or register, and can access the public view page. While logged in, a user can logout, and can access both the public view page and their own feed.

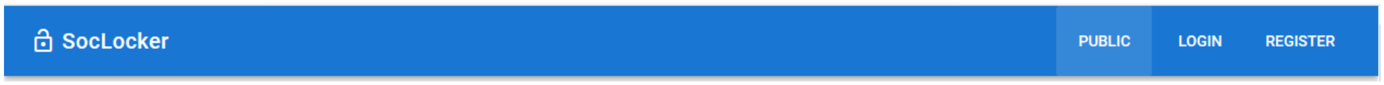


Fig. 16. Top Bar While Logged Out

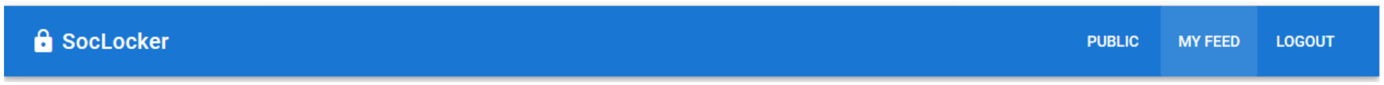


Fig. 17. Top bar While Logged In

8 Testing: Verification and Validation

It would, in theory, be possible to fully white box test the system proposed. However, as SocLocker serves only as a reference implementation, and would require extensive component integration testing with the number of systems which communicate, it is likely more time effective to black box test the finished system.

To fulfil this, a full system functionality test will be performed.

1) Account creation
2) Logging in
3) Logging out
4) Content creation
5) Reading ones own content
6) Reading a post created by another user
7) Editing content
8) Reading ones own edited content
9) Reading content edited by another user

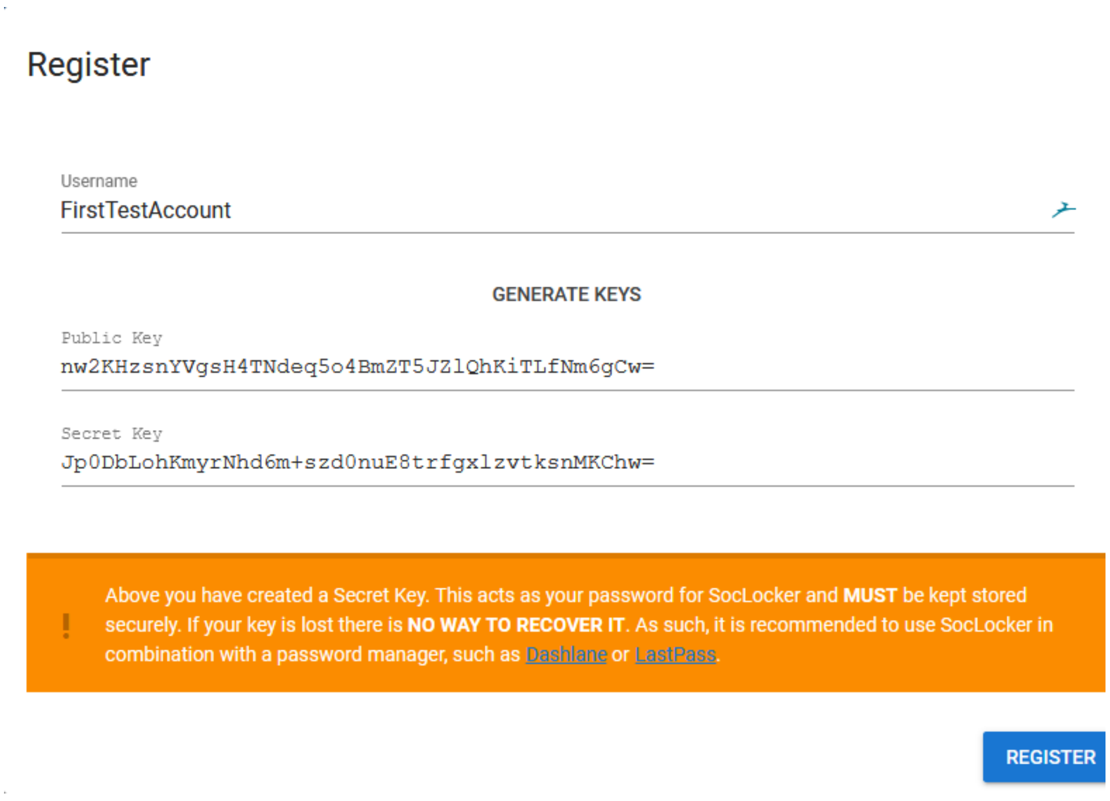
By verifying that ones own content, and another users content can be read, multi-user content is tested: as it functions as content that is sent to both oneself and another.

After these initial functionality tests are carried out, a number of trivial security flaws will be attempted to be exploited.

10) Creating content as another user
11) Accessing content one is not authorised to access

8.1 Account Creation

To create an account, the register button is used in the top right of the interface. This brings up the following dialogue.


Register

Username
FirstTestAccount

GENERATE KEYS

Public Key
nw2KHzsnYVgsH4TNdeq5o4BmZT5JZlQhKiTLfNm6gCw=

Secret Key
Jp0DbLohKmyrNhd6m+szd0nuE8trfgxlzvtksnMKChw=

Above you have created a Secret Key. This acts as your password for SocLocker and **MUST** be kept stored securely. If your key is lost there is **NO WAY TO RECOVER IT**. As such, it is recommended to use SocLocker in combination with a password manager, such as Dashlane or LastPass.

Fig. 18. Account Creation Dialogue

Creating an account, `FirstTestAccount`, with the private key it generates for us:

`Jp0DbLohKmyrNhd6m+szd0nuE8trfgxlzvtksnMKChw=`.

We are then given the following alert to provide us a final opportunity to store our record in a password generator.

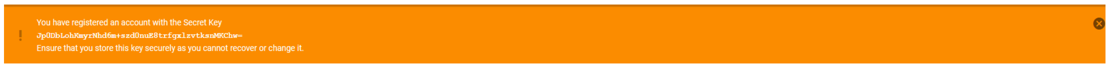
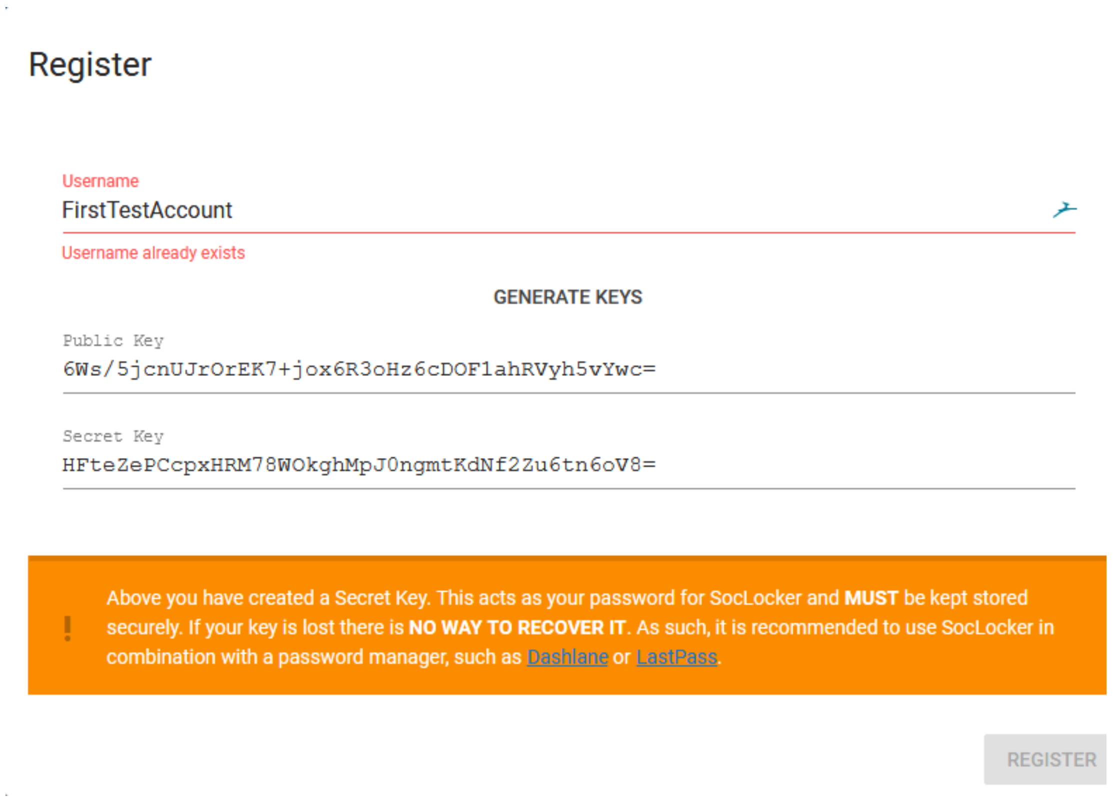

You have registered an account with the Secret Key
Jp0DbLohKmyrNhd6m+szd0nuE8trfgxlzvtksnMKChw=
Ensure that you store this key securely as you cannot recover or change it.

Fig. 19. Account Creation Secret Key Alert

We should not be able to create an account on an already existing username. The interface rejects this.


Register

Username
FirstTestAccount
Username already exists

GENERATE KEYS

Public Key
6Ws/5jcnUJrOrEK7+jox6R3oHz6cDOF1ahRVyh5vYwc=

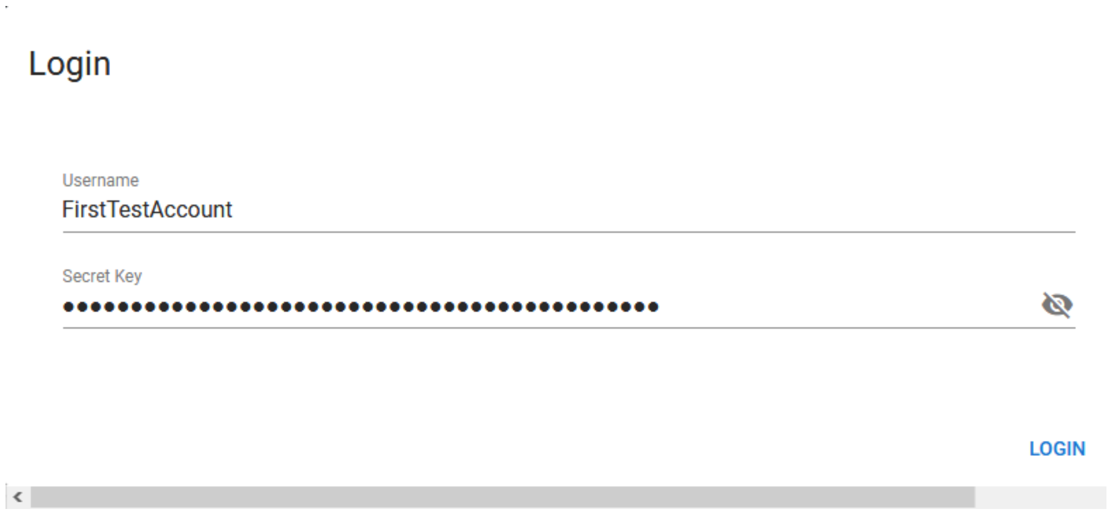
Secret Key
HFteZePCcpxHRM78WOkghMpJ0ngmtKdNf2Zu6tn6oV8=

Above you have created a Secret Key. This acts as your password for SocLocker and **MUST** be kept stored securely. If your key is lost there is **NO WAY TO RECOVER IT**. As such, it is recommended to use SocLocker in combination with a password manager, such as Dashlane or LastPass.

Fig. 20. Account Creation With Pre-Existing Username

8.2 Logging In

To log in, the login button in the top right of the interface can be used. This brings up the following dialogue.


Login

Username
FirstTestAccount

Secret Key
●●

Fig. 21. Login Dialogue

When logged in, the interface switches to the users feed, confirming the log in.

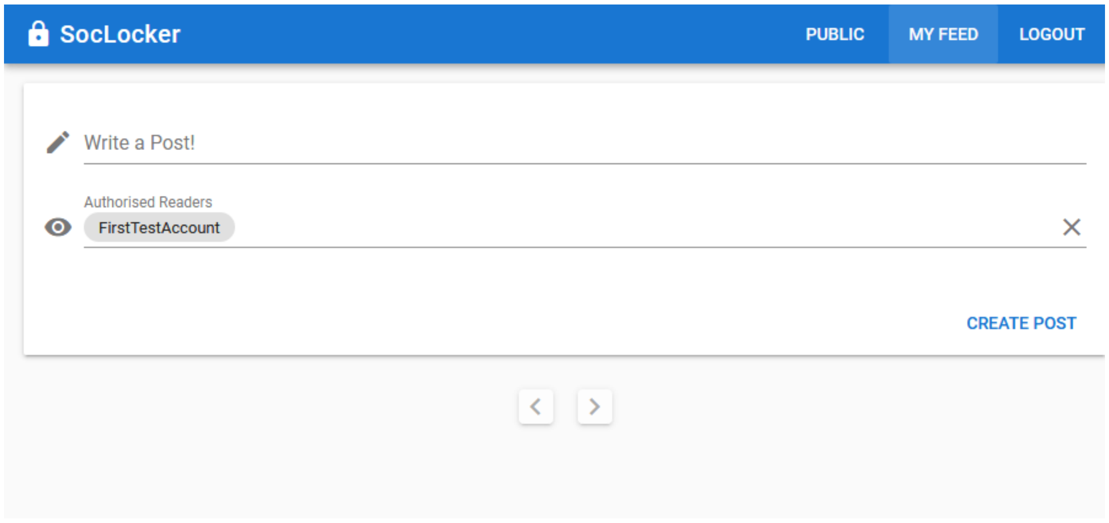


Fig. 22. User Feed Interface

It should not be possible to log in without the correct password.

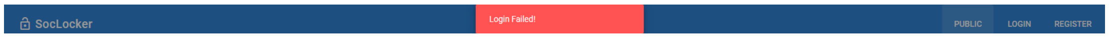
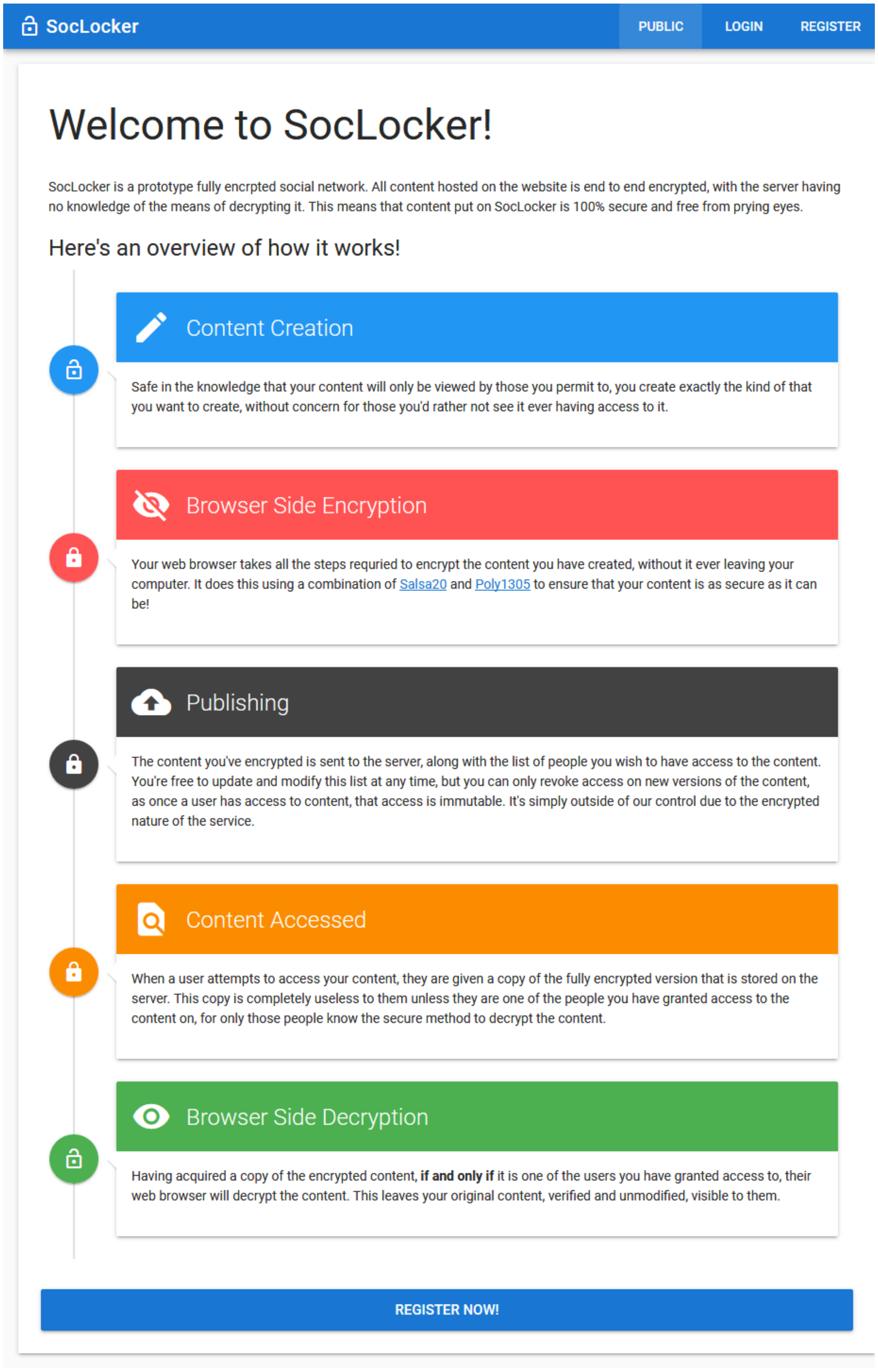


Fig. 23. Invalid Login With Incorrect Password

Attempting to do so results in a 'Login Failed' snack-bar being displayed.

8.3 Logging Out

Clicking the Logout button in the top right works as intended. Instantly bringing the user back to the main global page with information about the service.



Welcome to SocLocker!

SocLocker is a prototype fully encrpted social network. All content hosted on the website is end to end encrypted, with the server having no knowledge of the means of decrypting it. This means that content put on SocLocker is 100% secure and free from prying eyes.

Here's an overview of how it works!

Content Creation

Safe in the knowledge that your content will only be viewed by those you permit to, you create exactly the kind of that you want to create, without concern for those you'd rather not see it ever having access to it.

Browser Side Encryption

Your web browser takes all the steps requried to encrypt the content you have created, without it ever leaving your computer. It does this using a combination of Salsa20 and Poly1305 to ensure that your content is as secure as it can be!

Publishing

The content you've encrypted is sent to the server, along with the list of people you wish to have access to the content. You're free to update and modify this list at any time, but you can only revoke access on new versions of the content, as once a user has access to content, that access is immutable. It's simply outside of our control due to the encrypted nature of the service.

Content Accessed

When a user attempts to access your content, they are given a copy of the fully encrypted version that is stored on the server. This copy is completely useless to them unless they are one of the people you have granted access to the content on, for only those people know the secure method to decrypt the content.

Browser Side Decryption

Having acquired a copy of the encrypted content, **if and only if** it is one of the users you have granted access to, their web browser will decrypt the content. This leaves your original content, verified and unmodified, visible to them.

Fig. 24. The Global Page After Logging Out

8.4 Content Creation

Content can be created by entering it in the post creation box, and then submitting it with a list of accounts to add. For the purposes following tests, two additional test accounts have been created: `SecondTestAccount` and `ThirdTestAccount`.

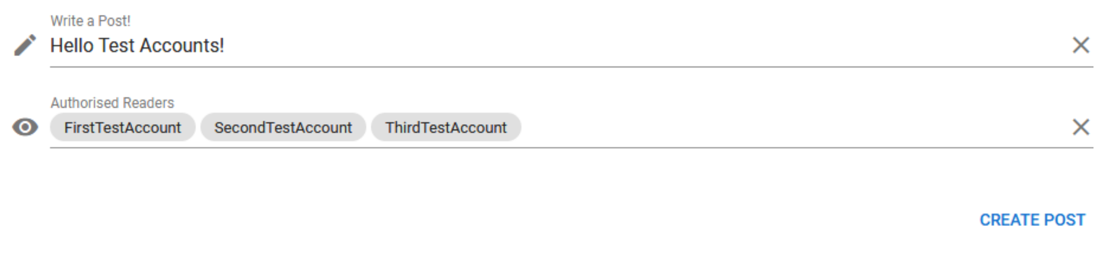
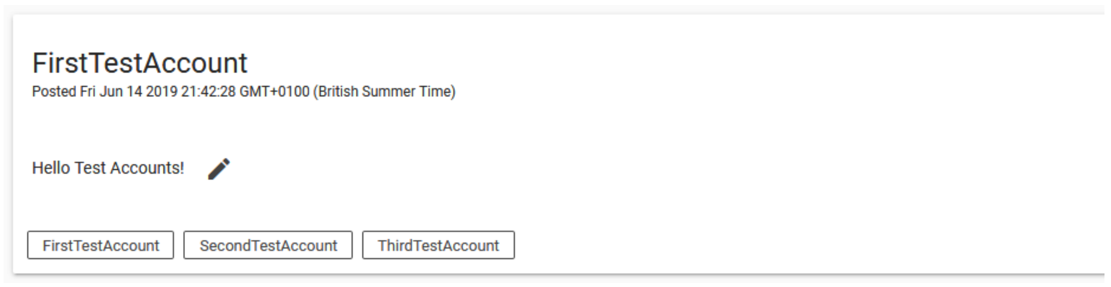


Fig. 25. Content Creation

No error is given off when the content is created, and the following test demonstrates this ones success.

8.5 Reading Ones Own Content

Once content is created, it is possible to read ones own content.



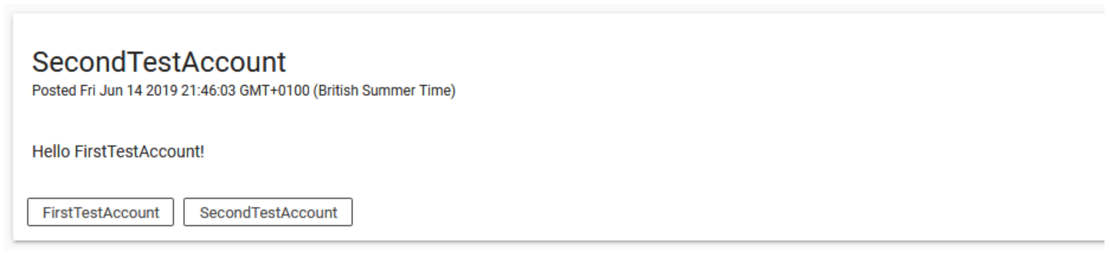
FirstTestAccount
Posted Fri Jun 14 2019 21:42:28 GMT+0100 (British Summer Time)

Hello Test Accounts!

Fig. 26. Reading Own Content

8.6 Reading Content Created By Another User

If content is created by a user, other users are also able to read it. By creating content from `SecondTestAccount`, we can read it while logged in as `FirstTestAccount`.


SecondTestAccount
Posted Fri Jun 14 2019 21:46:03 GMT+0100 (British Summer Time)

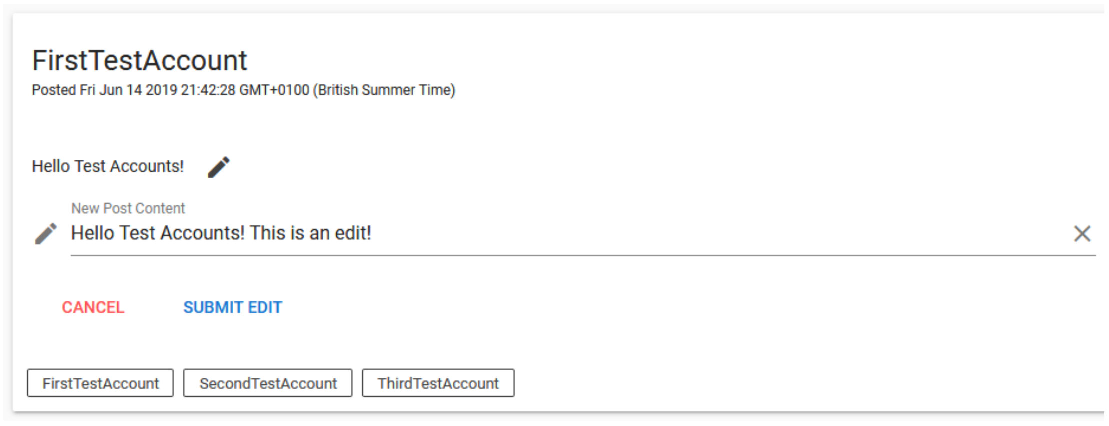
Hello FirstTestAccount!

Fig. 27. Reading Other User Created Content

It is trivial that content created that is not intended for a user is not seen by that user, as a high volume of test messages exist on the system from its early testing phases which are not documented here and are not visible to `FirstTestAccount`.

8.7 Editing Content

Content can be edited by clicking the pencil icon next to text in a post that a user has created.


FirstTestAccount
Posted Fri Jun 14 2019 21:42:28 GMT+0100 (British Summer Time)

Hello Test Accounts!

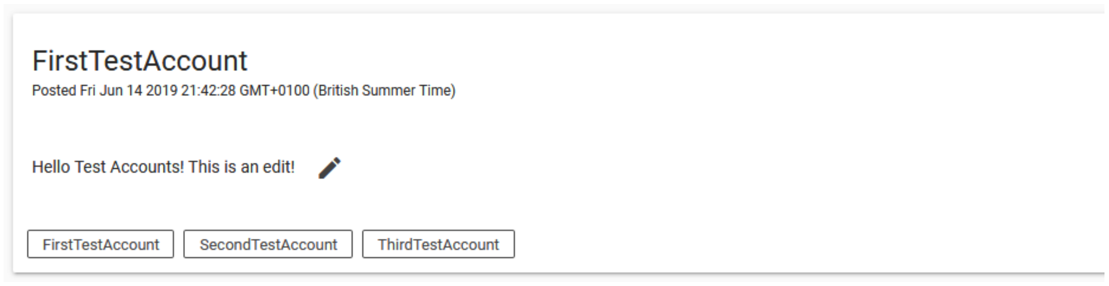
New Post Content
Hello Test Accounts! This is an edit!

Fig. 28. Editing Content

This edit is shown to be successful by the following test.

8.8 Reading Ones Own Edited Content

Once an edit has been made, the edit is accurately reflect in the content.

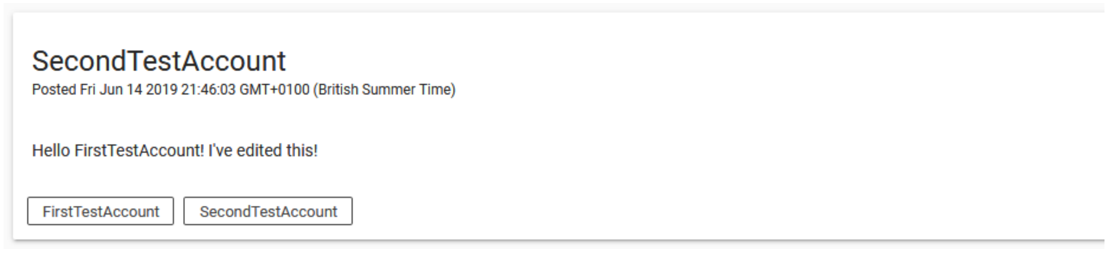

FirstTestAccount
Posted Fri Jun 14 2019 21:42:28 GMT+0100 (British Summer Time)

Hello Test Accounts! This is an edit!

Fig. 29. Reading Edited Content

8.9 Reading Content Edited By Another User

If another user edits content `FirstTestAccount` has access to, this change is accurately reflected in the feed.


SecondTestAccount
Posted Fri Jun 14 2019 21:46:03 GMT+0100 (British Summer Time)

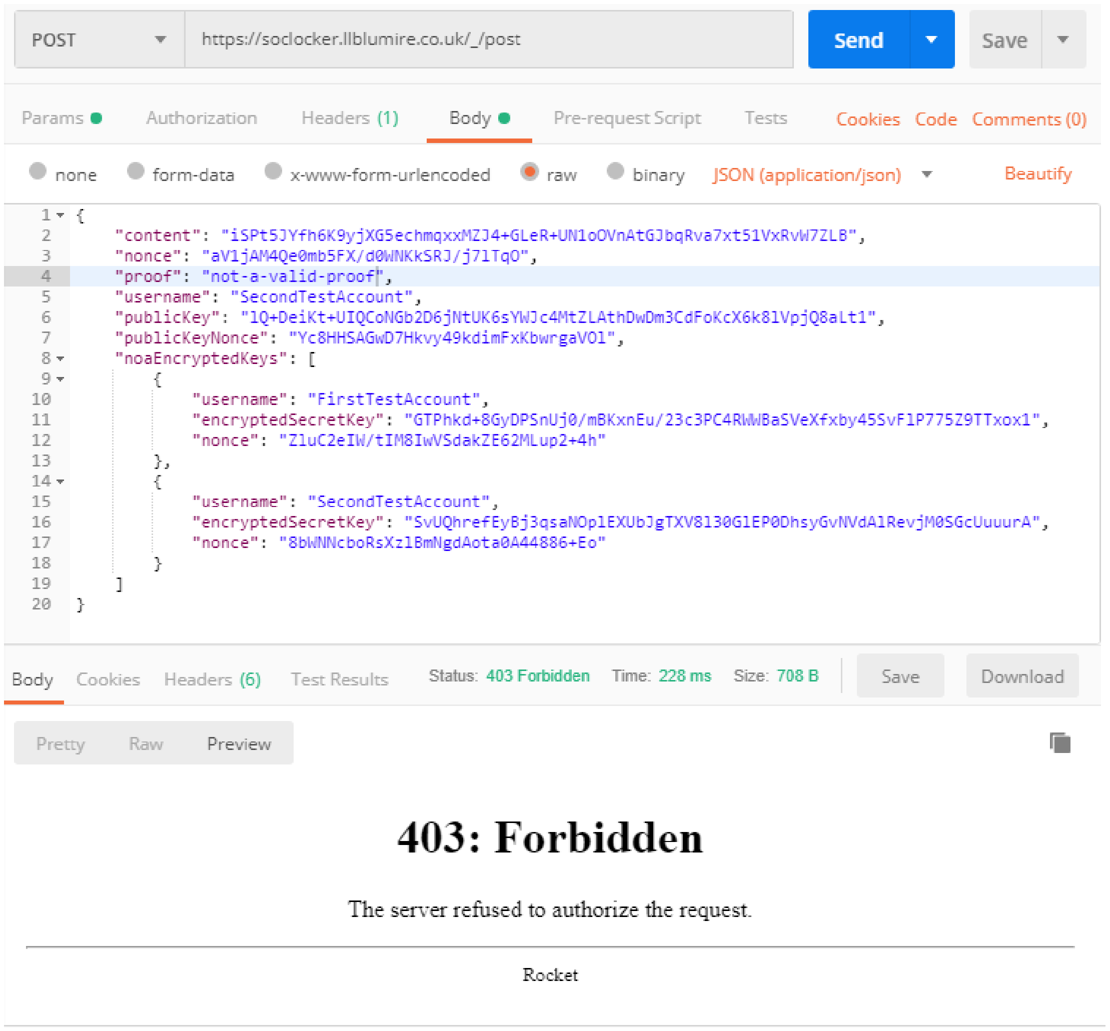
Hello FirstTestAccount! I've edited this!

Fig. 30. Reading Content Edited By Another User

8.10 Creating Content As Another User

The content creation API rejects a message without a valid proof.

To demonstrate this, let us attempt to post the message 'Hello World, You've Been Hacked!' as if we were `SecondTestAccount`, using only the information available to `FirstTestAccount`.



```
{
    "content": "iSPt5JYfh6K9yjXG5echmqxxMZJ4+GLeR+UN1oOVnAtGJbqRva7xt51VxRvW7ZLB",
    "nonce": "aV1jAM4Qe0mb5FX/d0WNKkSRJ/j71TqO",
    "proof": "not-a-valid-proof",
    "username": "SecondTestAccount",
    "publicKey": "1Q+DeiKt+UIQCoNGb2D6jNtUK6sYWJc4MtZLAthDwDm3CdFoKcX6k81VpjQ8aLt1",
    "publicKeyNonce": "Yc8HHSAGwD7Hkvy49kdimFxKbwrgaVOl",
    "noaEncryptedKeys": [
        {
            "username": "FirstTestAccount",
            "encryptedSecretKey": "GTPhkd+8GyDPSnUj0/mBKxnEu/23c3PC4RWWBaSVeXfxby45SvF1P775Z9TTxox1",
            "nonce": "Z1uC2eIW/tIM8IwVSdakZE62MLup2+4h"
        },
        {
            "username": "SecondTestAccount",
            "encryptedSecretKey": "SvUQhrefEyBj3qsaNOp1EXUbJgTXV8130G1EP0DhsyGvNVdA1RevjM0SGcUuuurA",
            "nonce": "8bWNNcboRsXz1BmNgdAota0A44886+Eo"
        }
    ]
}
```

Status: 403 Forbidden   Time: 228 ms   Size: 708 B

403: Forbidden

The server refused to authorize the request.

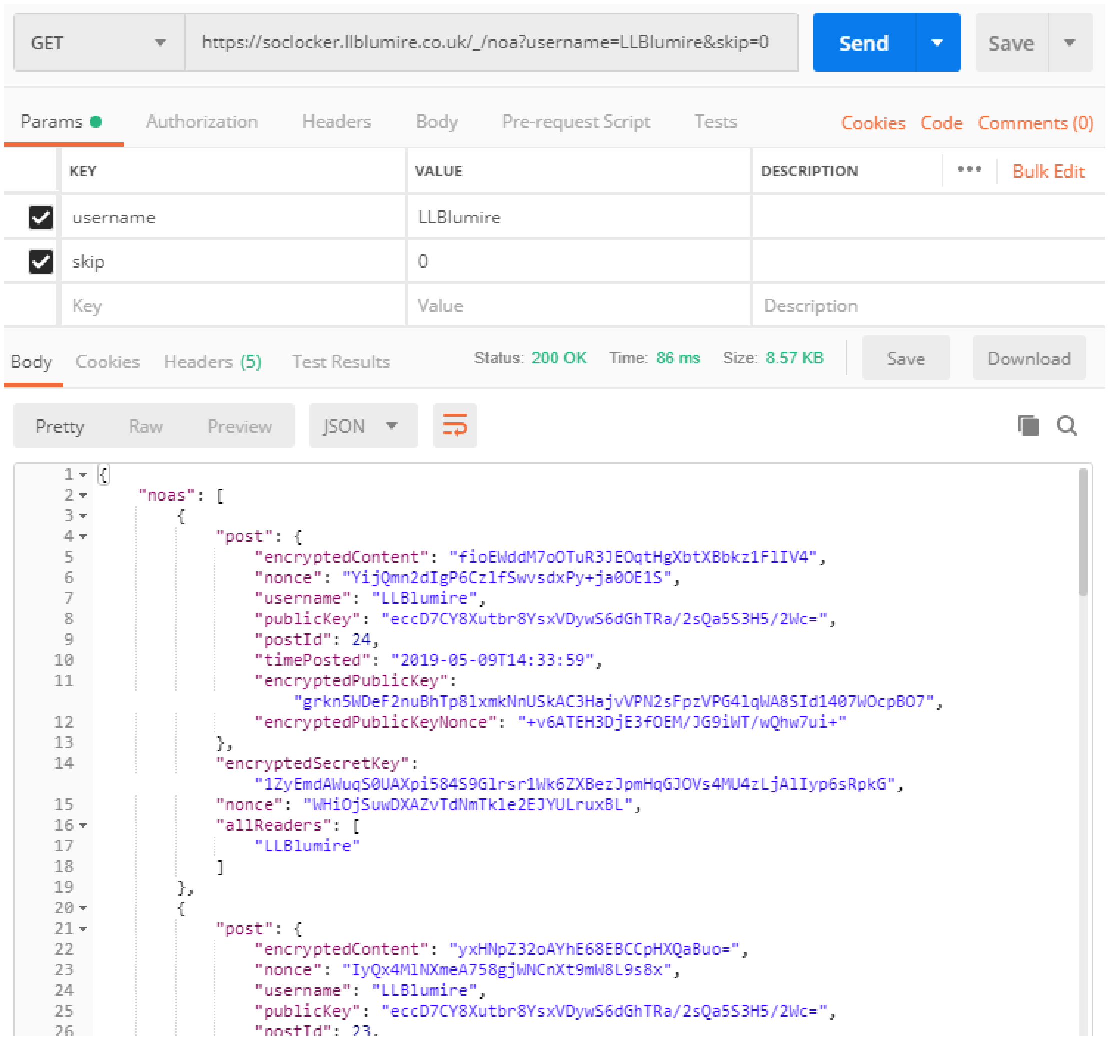
Rocket

Fig. 31. Creating Content As Another User

The server rejects the invalid proof, as desired. This demonstrates that the authentication layer works as intended, thus any other authentication layer dependent API will be equally secure, giving validation that the specification provides security from editing content that was created by other users as well.

8.11 Accessing Content One Is Not Authorised To Access

It is possible to request the content of any user. As demonstrated here requesting the posts accessible to a live data account: `LLBlumire`.


GET https://soclocker.llblumire.co.uk/_/noa?username=LLBlumire&skip=0

KEY	VALUE	DESCRIPTION
username	LLBlumire	
skip	0	

Status: 200 OK Time: 86 ms Size: 8.57 KB

```
{
    "noas": [
        {
            "post": {
                "encryptedContent": "fioEWddM7oOTuR3JEOqtHgXbtXBbkz1F1IV4",
                "nonce": "YijQmn2dIgP6Cz1fSwvsdxPy+ja0OE1S",
                "username": "LLBlumire",
                "publicKey": "eccD7CY8Xutbr8YsxVDywS6dGhTRa/2sQa5S3H5/2Wc=",
                "postId": 24,
                "timePosted": "2019-05-09T14:33:59",
                "encryptedPublicKey":
                    "grkn5WDeF2nuBhTp8lxmkNnUSkAC3HajvVPN2sFpzVPG41qWA8SId1407WOcpBO7",
                "encryptedPublicKeyNonce": "+v6ATEH3DjE3fOEM/JG9iWT/wQhw7ui+"
            },
            "encryptedSecretKey":
                "1ZyEmdAWuqS0UAXpi584S9Glrsr1Wk6ZXBezJpmHqGJOVs4MU4zLjAlIyp6sRpkG",
            "nonce": "WHiOjSuwDXAZvTdNmTkle2EJYULruxBL",
            "allReaders": [
                "LLBlumire"
            ]
        },
        {
            "post": {
                "encryptedContent": "yxHNpZ32oAYhE68EBCCpHXQaBuo=",
                "nonce": "IyQx4MlNXmeA758gjWNCnXt9mW8L9s8x",
                "username": "LLBlumire",
                "publicKey": "eccD7CY8Xutbr8YsxVDywS6dGhTRa/2sQa5S3H5/2Wc=",
                "postId": 23,
```

Fig. 32. Acquiring Content Accessible To Another User

However it is impossible to decrypt the data that was provided to that user without knowledge of their secret key, in order to decrypt the `encryptedSecretKey` for the content. It does however show that who has shared content with others is public information.

9 Discussion: Contribution and Reflection

Given the success of each test presented in Section 8, and the fulfilment of the specification provided by Section 2, it has been demonstrated that it is possible to construct a cryptographically secure social network that values user privacy and security above anything else.

SocLocker provides a full working implementation of the algorithms and cryptographic techniques discussed in this paper. It enables securely client-side encrypted content to be centrally stored, and accessed in a fashion congruent with social media. By integrating into password management software it allows for a user experience that is equivalent to other social networks and entirely non-disruptive.

The tests performed and attempts at system penetration demonstrate the relative strength of the system: which is based in its underlying mathematical formulation. By having security by mathematics rather than security by design or security by architecture, a higher level of information data security is attained.

There are a number of limitations to the project. Notably, the lack of common social media features. SocLocker lacks a direct messaging system, or a reply or content threading system common on social media. It also supports only one type of content: text.

These limitations are of questionable severity, as each could be resolved and implemented on top of the existing framework and server architecture. Messages starting with a token targetting a message ID could be interpreted by a client as replies. Messages starting with a token targetting a user ID could be interpreted by a client as direct messages. Messages starting with tokens representing other file formats could be interpreted as different media sources. All of this is possible without any server architecture changes, simply by providing a more advanced client than the minimum demonstration created for SocLocker.

Another limitation is the fact that the interaction graph is public. It is not trivially accessible as it would require individually harvesting the feeds of every user, and no full list of users is publicly available. It is however possible that a full graph of user interactions might be constructed, which is a privacy violation some users might not tolerate. It is easily circumvented however, as there is no requirement that a username be publicly associated with any person so as to reveal their identity.

As a personal learning experience, this project has highlighted to me my personal flaws in time management on large projects—something I'll need to work on in the future. This has posed the biggest difficulty throughout the entire project, from it's inception, to it's restart, to it's final conclusion. Were I to do it all again I would start with a more attainable specification, rather than start with something large and ambitious and eventually refine it down. That is not to say that the project as it is now is simple, but instead that it fits into a more rigidly definable structure. I've learnt that defining things in a mathematically sound and provable way and then abstracting them into a computer science focused system is the general sphere of computer science that I enjoy, and is something that I intend to pursue going forward in my academic career.

10 Social, Legal, Health & Safety, and Ethical Issues

10.1 Encryption Law

The United Kingdom of Great Britain and Northern Ireland—herein UK—does not have direct law preventing the usage of cryptography and the transmission of encrypted content. This is commonplace worldwide as the restricting of encrypted content would seriously harm the security and banking industries. The United States of America—herein USA—is notable for having some restrictions dating back to World War II where laws were implemented that restrict the exporting of cryptographic content, and the open source publishing of cryptographic programs. Similar restrictions to those in the USA apply in the French Republic, but they are heavily relaxed for relations with the European Union.

There is one significant law relating to cryptography within the UK, and that is the Regulation of Investigatory Powers Act 2000, or RIPA[10]. Part three of this report requires disclosure of cryptographic keys to government representatives without the use of a court order, against penalty of jail time. As user content is not protected by a private key known to the system, this creates no legal issue for SocLocker or other social media services built on the same technology.

RIPA does provide an immediate ethical benefit to the system presented however, as it allows regulatory powers to enforce ethical usage by system users in places where unethical and illegal usage occurs.

10.2 Ethical Use

Though Encryption and the hosting of Encrypted Content is by no means illegal, the design space of the social website outlined by this document presents a high number of ethical and social issues.

The first and foremost, can be plainly seen by taking the nature of the website in summary: "A service where pseudo-anonymous users can share content with a finite and small set of other pseudo-anonymous users in an entirely secure fashion."

While this is the intended design and makes the service ideal (and far more suited than other services which the following often occur over) for important business conversations, private discussions, and political negotiations. It also opens the gateway for piracy release groups, extreme illegal pornography, and terrorism: all things which operate within the confines of a small group sharing content they require to be secure and inaccessible to those legally authorised to penalise them for doing so.

This poses the primary ethical and social issue that is present within this form of end-to-end encrypted social media.

This issue cannot be completely resolved, as it is inherent to the design of the system. However, it can be mitigated and users can be given the highest opportunity possible to facilitate appropriate legal reaction to such content.

This is achieved through the encryption forwarding system. In a traditional end-to-end encrypted messaging system, revealing the content of a message in a verifiable fashion would require a user to reveal their own secret key, thus potentiality incriminating them for other illegal content you have accessed or distributed. Within the key forwarding system used, it is possible for a user to reveal the content of only a single message: thus only incriminating the sender and the other recipients, without further incriminating themselves on account any other potential content accessible through their secret key.

10.3 The Right To Be Forgotten

The European Unions General Data Protection Regulation gives a right to be forgotten in Article 17. This means that it must be possible for a user to erase themselves from the system. For this to be achieved, it is possible for a user to fully edit each of their posts, giving zero comment and erasing their content from the system. This does not remove them from the graph of content which has been provided, and a record is kept that they provided content to someone as is partially required for legal reasons with respect to RIPA, but the nature of the content is lost not only to the system which did not know it in the first place, but to any other users accessing it who had not saved a local copy.

11 Conclusion and Future Improvements

The objective of this project was to demonstrate a strong mathematical model for a social media service in which all content would be encrypted between all users, without severely inconveniencing the user and requiring them to have an understanding of the cryptographic techniques at play.

Through the mathematical definitions provided in Section 5 and the Implementations provided through Sections 6 and 7, a robust service has been constructed that meets those requirements. It is not without its limitations, but it gives a strong foundation for a system that could be extended and built on top of in future to provide a more secure and private online social media experience.

To summarise what has been achieved in this project.

- The definition of a mathematical system for multiple-reader encryption which minimises the re-encryption needed for editing.
- The implementation of a full server implementation for a fully encrypted social network, including the user authentication layer, the content hosting, and a full database design for the system.
- The design and implementation of an intuitive to use social media system that should have high familiarity and that does not require foreknowledge of the encryption technologies used.
- A full and comprehensive black-box test of the system and its implementations.
- All of the above, in a way that is legally compliant with both the UKs RIPA, the EUs GDPR, and provides a basis for reasonable means to restrict unethical usage.

The system proposed learns heavily from the design space that has been implemented for blockchain cryptographic technologies, but uses centralisation to reduce complexity and barrier to entry and to provide permanent mutability (as required by law as discussed in Section 10). Further learning could be done with this to extend the system.

One extension builds on what is mentioned in 9 regarding the fact that the connectivity graph of the system is public, i.e. it is always possible to find out who has shared content with who.

A possible way to obfuscate this could be taken from block-chain technology as a further privacy improvement to the system: taking elements of the Zero Knowledge Proof system and using that to enhance the privacy of the system at hand. A cryptocurrency called Zcash uses a technology called zk-SNARK which stands for "Zero-Knowledge Succinct Non-Interactive

Argument of Knowledge" to allow for transactions to occur without public disclosure of the sender, receiver, or the amount of money sent. This is revolutionary, as blockchain cryptocurrency is based largely on the premise that every user can verify at all times the exact amount any other user has access to in order to proove that a user has the money they are claiming to be able to send. A similar technique to the zk-SNARK could be used in order to obfuscate similarly the sender, receivers, and content of a message, while still having that message be accessible to those that are supposed to have received it.

A simpler version of this could be implemented by requiring the usage of the authentication layer to access the posts that have been made available to a user. This would not secure the backend of the system however, which is assumed to be public to ensure that administrators are not a potential security risk. The only exception to this is the keys that allow the authentication layer to function, which are single use and so if compromised would give limited vulnerability. Without having access to a users actual private key the extent of the damage a hacker or rogue administrator could do would be to post garbage non-decryptable data pretending to be a user they are not, or to do the same overwriting existing content. Of course, a rogue administrator could do this anyway simply by editing the database.

In conclusion: with only a single significant privacy infraction that many could forgive—and which could be solved with future extensions to the system taking advantage of advanced zero-knowledge computation techniques—and no significant understanding of the underlying cryptographic techniques being required to use the system, the project has been a definite success.

References

[1] Z. Whittaker, “Facebook now says its leak affected ’millions’ of instagram users techcrunch,” Apr 2019. [Online]. Available: https://techcrunch.com/2019/04/18/instagram-password-leak-millions/

[2] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978. [Online]. Available: http://doi.acm.org/10.1145/359340.359342

[3] “Keybase crypto documents.” [Online]. Available: https://keybase.io/docs/crypto/overview

[4] D. J. Bernstein, “Nacl: Networking and cryptography library.” [Online]. Available: https://nacl.cr.yp.to/

[5] S. Nakamoto *et al.*, “Bitcoin: A peer-to-peer electronic cash system,” *bitcoin.org*, 2008.

[6] D. J. Bernstein, “The salsa20 family of stream ciphers,” in *New Stream Cipher Designs*, M. Robshaw and O. Billet, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 84–97. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68351-3_8

[7] ——, “A state-of-the-art message-authentication code.” [Online]. Available: http://cr.yp.to/mac.html

[8] ——, “A state-of-the-art diffie-hellman function.” [Online]. Available: https://cr.yp.to/ecdh.html

[9] D. J. Bernstein, T. Lange, and P. Schwabe, “The security impact of a new cryptographic library,” in *Progress in Cryptology – LATINCRYPT 2012*. Springer Berlin Heidelberg, 2012, pp. 159–176. [Online]. Available: https://doi.org/10.1007/978-3-642-33481-8_9

[10] “Regulation of investigatory powers act 2000 part iii,” Jul 2000. [Online]. Available: https://www.legislation.gov.uk/ukpga/2000/23/part/III

APPENDIX A

PROJECT INITIATION DOCUMENT

Individual Project (CS3IP16)

Department of Computer Science
University of Reading

Project Initiation Document

PID Sign-Off

Student No.	**25010846**
Student Name	**Lucille Lillian Blumire**
Email	**l.blumire@student.reading.ac.uk**
Degree programme (BSc CS/BSc IT)	**BSc CS**
Supervisor Name	
Supervisor Signature	
Date	

SECTION 1 – General Information

Project Identification

1.1	**Project ID** (as in handbook)
	OWN
1.2	**Project Title**
	Securing of Social Content with non-disruptive cryptography
1.3	**Briefly describe the main purpose of the project in no more than 25 words**
	A system enabling users to manage cryptographically secure social posts without understanding the underlying cryptographic principles.

Student Identification

1.4	**Student Name(s), Course, Email address(s)** e.g. Anne Other, BSc CS, a.other@student.reading.ac.uk
	Lucille Lillian Blumire, BSc CS, l.blumire@student.reading.ac.uk

Supervisor Identification

1.5	**Primary Supervisor Name, Email address** e.g. Prof Anne Other, a.other@reading.ac.uk
1.6	**Secondary Supervisor Name, Email address** Only fill in this section if a secondary supervisor has been assigned to your project

Company Partner (only complete if there is a company involved)

1.7	**Company Name**
1.8	**Company Address**
1.9	**Name, email and phone number of Company Supervisor or Primary Contact**

SECTION 2 – Project Description

2.1	**Summarise the background research for the project in about 400 words. You must include references in this section but don't count them in the word count.**
	Cryptography is becoming a more prevalent part of the modern world, with more people each year caring about how their data is handled. This has led to a number of laws being passed (such as the GDPR [1]) to ensure people have more control over who is accessing their data, but many people still post things on social media in a more public capacity than they realise. Around 70% [2] of employers are investigating the things posted by their prospective employees, and there have been high profile cases of people being fired due to posts they made long ago being too public and against modern public tastes [3][4]. A means of securing messages cryptographically has existed for a long time, and much of the logic intended for this project is built off of public key cryptography [5]. Work has been done into using this for multi-key cryptography have been developed to allow for multiple groups of users to sign a single document. This has been largely developed and pushed forward by cryptocurrency research [6] though these have been prone to issues to do with relying on a centralized verification method [7]. Such techniques are also heavily based on custom signature generation and verification algorithms, while the aim of this project would be to develop something far more generic. Techniques for sharing secrets have been developed, such as the Diffie–Hellman key exchange [8], but there is little work when it comes to configurable, mutable, group based cryptography. [1] https://eugdpr.org/ [2] https://www.careerbuilder.com/advice/social-media-survey-2017 [3] https://www.hollywoodreporter.com/heat-vision/james-gunn-exits-guardians-galaxy-vol-3-1128786 [4] https://www.dailymail.co.uk/news/fb-5976069/JAMES-GUNNS-DELETED-TWEETS.html [5] https://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-standard-wp.pdf [6] https://doi.org/10.1007/11967668_10 [7] https://cointelegraph.com/news/parity-multisig-wallet-hacked-or-how-come [8] https://ee.stanford.edu/%7Ehellman/publications/24.pdf
2.2	**Summarise the project objectives and outputs in about 400 words.** These objectives and outputs should appear as tasks, milestones and deliverables in your project plan. In general, an objective is something you can do and an output is something you produce – one leads to the other.
	The project aims to create a system wherein a modelled social system (i.e. social media) can be used and contributed to in such a fashion that only selective users are able to read posts, in such a fashion where posts contain minimum redundancy in their encoding and can be trivially mutated, both in terms of the underlying content and the set of people granted access to it. The algorithm developed to do this should facilitate generic application over public key cryptosystems, rather than locking in to any specific cryptographic methods. In order to make the system non obtrusive, a user should be able to create their private key by a secure mnemonic, that will ensure the key is memorable to the user but also such that the mnemonic space uses the entirety of the key space fairly.

2.3	**Initial project specification - list key features and functions of your finished project.** Remember that a specification should not usually propose the solution. For example, your project may require open source datasets so add that to the specification but don't state how that data-link will be achieved – that comes later.
	• Generate secure, Memorable mnemonics for private keys. • Account creation based on a private and public key. • Login verification of this private key against a public key or username. • Ability to make a text post, public, and signed. • Ability to make a text post, private, readable by a single person. (A message) • Ability to make a text post, private, readable by a mutable group of people. (A group message) • Ability to edit any post made, and persist those edits to all users. • Ability to delete any post made, and have it become inaccessible short of users saving a copy locally.
2.4	**Describe the social, legal and ethical issues that apply to your project. Does your project require ethical approval? (If your project requires a questionnaire/interview for conducting research and/or collecting data, you will need to apply for an ethical approval)**
	This project should not require ethical approval. Laws may in the future be passed to limit the use of cryptography, but at present the project has no legal issues.
2.5	**Identify and lists the items you expect to need to purchase for your project. Specify the cost (include VAT and shipping if known) of each item as well as the supplier.** e.g. item 1 name, supplier, cost
	No required purchases
2.6	**State whether you need access to specific resources within the department or the University e.g. special devices and workshop**
	No required resources

SECTION 3 – Project Plan

3.1	**Project Plan** Split your project work into sections/categories/phases and add tasks for each of these sections. It is likely that the high-level objectives you identified in section 2.2 become sections here. The outputs from section 2.2 should appear in the Outputs column here. Remember to include tasks for your project presentation, project demos, producing your poster, and writing up your report.

Task No.	Task description	Effort (weeks)	Outputs
1	**Background Research**		
1.1	Public Key Cryptography Research	1	
1.2	Multisignature Research	2	
2	**Analysis and design**		
2.1	Analysis of Social Media Requirements	2	
2.2	Analysis of Encryption Algorithm Requirements	2	
2.3	Design of Mnemonic System	1	Mnemonic Key Generation Design
2.4	Design of Prototype Social Media	2	Social Media Design
	Design of Public Key Encryption Systems	2	Encryption System Design
3	**Develop prototype**		
3.1	Develop Prototype Social Media	3	Prototype Social Media Service
3.2	Implement Public Key Encryption Techniques	2	Ability to post private messages to the Service
3.3	Implement Mutability Techniques	3	Ability to mutate messages and reader permissions
4	**Testing, evaluation/validation**		
4.1	(Unit Tests Done Throughout Development)	0	
4.2	Encryption Security Testing	3	
5	**Assessments**		
5.1	Write Up Project Report	5	Project Report
5.2	Produce Project Poster	1	Poster
5.3	Demo Project	1	
TOTAL	**Sum of total effort in weeks**		

SECTION 4 - Time Plan for the proposed Project work

For each task identified in 3.1, please *shade* the weeks when you'll be working on that task. You should also mark target milestones, outputs and key decision points. To shade a cell in MS Word, move the mouse to the top left of cell until the curser becomes an arrow pointing up, left click to select the cell and then right click and select 'borders and shading'. Under the shading tab pick an appropriate grey colour and click ok.

START DATE: 01/01/2019 **<enter the project start date here>**

Project Weeks

Project stage	0-3 1/1	3-6 22/1	6-9 12/2	9-12 5/3	12-15 26/3	15-18 16/4	18-21	21-24	24-27	27-30	30-33	33-36	36-39
1 Background Research	▓						█	█	█	█	█	█	█
1.1 Public Key Cryptography Research	░						█	█	█	█	█	█	█
1.2 Multisignature Research	░						█	█	█	█	█	█	█
2 Analysis/Design		▓	▓				█	█	█	█	█	█	█
2.1 Analysis of Social Media Requirements		░					█	█	█	█	█	█	█
2.2 Analysis of Encryption Algorithm …		░					█	█	█	█	█	█	█
2.3 Design of Mnemonic System		░	░				█	█	█	█	█	█	█
2.4 Design of Prototype Social Media		░	░				█	█	█	█	█	█	█
3 Develop prototype.			▓	▓			█	█	█	█	█	█	█
3.1 Develop Prototype Social Media			░				█	█	█	█	█	█	█
3.2 Implement Public Key Encryption…				░			█	█	█	█	█	█	█
3.3 Implement Mutability Techniques							█	█	█	█	█	█	█
4 Testing, evaluation/validation				▓			█	█	█	█	█	█	█
4.2 Encryption Security Testing				░			█	█	█	█	█	█	█
5 Assessments				▓	▓	▓	█	█	█	█	█	█	█
5.1 Write up Project Report				░	░	░ 29/4	█	█	█	█	█	█	█
5.2 Write up Project Poster				░	░ 29/3		█	█	█	█	█	█	█
5.3 Demo Project				░	░	░ 26/4	█	█	█	█	█	█	█

RISK ASSESSMENT FORM

Assessment Reference No.		**Area or activity assessed:**	
Assessment date			
Persons who may be affected by the activity (i.e. are at risk)			

SECTION 1: Identify Hazards - *Consider the activity or work area and identify if any of the hazards listed below are significant (tick the boxes that apply).*

1.	Fall of person (from work at height)		**6.**	Lighting levels		**11.**	Use of portable tools / equipment		**16.**	Vehicles / driving at work		**21.**	Hazardous fumes, chemicals, dust		**26.**	Occupational stress	
2.	Fall of objects		**7.**	Heating & ventilation		**12.**	Fixed machinery or lifting equipment		**17.**	Outdoor work / extreme weather		**22.**	Hazardous biological agent		**27.**	Violence to staff / verbal assault	
3.	Slips, Trips & Housekeeping		**8.**	Layout , storage, space, obstructions		**13.**	Pressure vessels		**18.**	Fieldtrips / field work		**23.**	Confined space / asphyxiation risk		**28.**	Work with animals	
4.	Manual handling operations		**9.**	Welfare facilities		**14.**	Noise or Vibration		**19.**	Radiation sources		**24.**	Condition of Buildings & glazing		**29.**	Lone working / work out of hours	
5.	Display screen equipment		**10.**	Electrical Equipment		**15.**	Fire hazards & flammable material		**20.**	Work with lasers		**25.**	Food preparation		**30.**	Other(s) - specify	

SECTION 2: Risk Controls - *For each hazard identified in Section 1, complete Section 2.*

Hazard No.	Hazard Description	Existing controls to reduce risk	**Risk Level** (tick one)			Further action needed to reduce risks *(provide timescales and initials of person responsible)*
			High	Med	Low	
Name of Assessor(s)			**SIGNED**			
Review date						

Health and Safety Risk Assessments – continuation sheet

Assessment Reference No	
Continuation sheet number:	

SECTION 2 continued: Risk Controls

Hazard No.	Hazard Description	Existing controls to reduce risk	**Risk Level** (tick one)			Further action needed to reduce risks
			High	Med	Low	*(provide timescales and initials of person responsible for action)*

Name of Assessor(s)		**SIGNED**
Review date		

Appendix B

Logbook

Date	Worked On	Hours Worked
15/03/2019	Development Environment Setup	7
16/03/2019	Server initial authentication system (auth endpoint only).	6
17/03/2019	Client authentication system (login screen).	3
19/03/2019	Initial mathematical proposal for solution.	6
20/03/2019	Refinement to mathematical proposal.	5
01/05/2019	Full code refactor for vue+vuetify architecture	
04/05/2019	Server post read and write functionality. Client write functionality. Notices of access.	7
09/04/2019	Client post read functionality.	10
11/04/2019	Mathematical formalisation of system.	5
12/04/2019	Mathematical formalisation of system.	8
14/04/2019	Server post edit functionality. Client refactor to make use of async/await.	4
15/04/2019	Client post edit functionality. Reimplementation of authentication system (usable on any endpoint). Server post security improvements.	6
16/04/2019	Project report restructuring and appendix collating.	1
03/05/2019	Writing Glossary of Terms and Abbreviations	1
08/05/2019	Documenting Server Implementation Sequences	5
09/05/2019	Documenting Server Implementation API Calls	3
10/05/2019	Documenting Client Side Implementation	4
11/05/2019	Writing Technical Specification	1
13/06/2019	Writing the Introduction, Solution Approach, and Encryption Law	8
14/06/2019	Writing the Test Plan and Beginning Testing	5
15/06/2019	Writing the Discussion, and Conclusion	8
16/06/2019	Writing the Literature Review	6

Total Hours Spent: 109.

It should be noted that that above hours spent represent direct time spent writing code, documentation, or this report. It does not include the countless hours spent considering the problem and its potential solutions passively throughout the course of the year.