COMP208 - Group Software Project
Ballmer Peak

M. Chadwick; Choi, S.F (AKA Leon); P. Duff; L. Prince; A.Senin; L. Thomas

May 9, 2014

Contents

Part I

Requirements

Chapter 1

Mission Statement

"to shift societal norms to a state wherein privacy is respected without caveat or justification"

In light of dissidents utilizing social networking websites such as Facebook and Twitter to organize protests, we feel that there is a need for an easy to use, encrypted communications platform with support for real-time and asynchronous communication between users.

Chapter 2

Mission Objectives

The proposed project (Turtlenet) is a simple, privacy oriented social network, which demands zero security or technical knowledge on behalf of its users. In order to ensure security and privacy in the face of nation state adversaries the system must be unable spy on its users even if it wants to or server operators intend to.

We feel that obscuring the content of messages isn't enough, because suspicion may, and often does, fall upon people not for what they say, but to whom they are speaking[**trackingFriends**]. Our system will therefore not merely hide the content of messages, but the recipient of messages too. Hiding the fact that an IP address sent a message is out of scope, but hiding which user/keypair did so is in scope, as is which IP/user/keypair received the message and the content of the message. It is important to hide the recipient of the message, because otherwise they may be unfairly targeted[**droneCellTracking**] if they use our services to communicate with the wrong people on a phone which is later identified, or they may merely be 'selected' for spying and harassment [**greenwaldAnnoying**].

We feel that current tools have significant usability problems, as was recently made starkly clear when Glenn Greenwald, a reporter of the guardian, was unable to work with Edward Snowden because he found GPG to be "too annoying" to use.

> "It's really annoying and complicated, the [email] encryption software" - Glenn Greenwald [**greenwaldAnnoying**]

While there exist many tools for hiding what you are saying, relatively few seek to hide who talks with whom, and those which do often implement it merely as a proxy, or seek to provide convenience over security.

The system is to have strict security measures implemented. It is able to encrypt messages with the use of RSA and AES. The only way for the other user to decrypt the data is if it was encrypted

using their public key; which is given from the recipient to the sender via whichever medium he prefers, e.g. email. We will also allow users to transmit public keys as QR codes, for ease of use.

The system will provide a platform for people to securely communicate, both one-to-one and in groups. Users will be able to post information to all of their friends, or a subset of them as well as sharing links and discussing matters of interest.

The following are our main design goals. Please note that the system is designed with axiom that the server operators are unjust, seeking to spy on potential users, and able to modify the source for the server.

- Strong cryptography protecting the content of messages
- Make it an impossible task to derive, from the information the server has or is able to collect, which users send a message to which users
- Make it an impossible task to derive, from the information the server has or is able to collect, which users receive a message at all
- Transmission of public key is easy, and doesn't require knowing what a public key is
- Be intuitive and easy to use, prompting the user when required
- Provide a rich social network experience, so as to draw regular members and drive up network diversity

The server operator will have access to the following information:

- Which IP uploaded which message (although they will be ignorant of its content, type, recipient, and sender)
- Which IPs are connecting to the server as clients (but not what they view, whom they talk with, or whether they receive a message at all)
- What times a specific IP connects [1]

A third party logging all traffic between all clients and a server will have access to what IPs connect to the server, and whether they upload or download information [2]

The benefits we feel this system provides over current solutions are:

- Server operators can not know who talks with whom
- Server operators can not know the content of messages

[1] While this will aid in tying an IP address to a person, it is deemed acceptable because it is not useful information unless the persons private key is compromised.

[2] size correlation attacks could be used here if the message content is known

- Server operators can not know which message is intended for which user
- Server operators can not know who is friends with whom

In order to ensure nobody can tell who is talking with whom, we will base our security model on the idea of shared mailboxes, as seen in practice at alt.anonymous.messages [3]. In this model one posts a message by encrypting it using the public key of the recipient, and posting it in a public location. In this model, one reads a message by downloading all messages from that location, and attempting to decrypt them all using ones private key. Our protocol will build atop this simple premise, and the the server will be a mere repository of messages, the real work occurring wholly in the client.

[3] https://groups.google.com/forum/#!forum/alt.anonymous.messages

Chapter 3

Project Target

A project of this scope has a rather specific target in sight. Due to its encrypted nature, Turtlenet can act as a form of anonymity between users who would otherwise be targeted by governments and/or institutions opposed to them. Countries such as China[**chinafirewall**] and a majority of the middle east[**libyaEgypt**] have recently seen negative press due to their persecution of individuals whom disagree with the ruling regime, such software would allow said individuals safety from what the wider world views as acceptable.

Large multinational defence corporations (e.g. IBM, Thales, BAE) might also find Turtlenet useful, as it would allow for a secure communication tool between employees in an office. It could also potentially be used outside a company firewall to send messages securely between offices across much larger distances. Corporations such as defence contractors often hold security in the highest regard, and such a project would match their needs well.

A more likely recipient of this system however, is society itself, as we have decided to waive our copyright granted monopoly. Should another group decide to embark on a similar project, they will have access to this project, to act as a baseline for their own work. See Appendix **??**.

Chapter 4

Threat Model

When designing a system in which security is a significant aspect, it helps to define clearly exactly what adversaries are anticipated. In this section we will describe a hypothetical adversary (hereafter 'the adversary') against whom we will protect our users.

The adversary will be granted all powers available to all conceivable attackers, such that no collusion of attackers may overcome our security (should it work for any given considered attacker).

The following individual attackers are considered, those attackers excluded are excluded on the basis that their abilities are a subset of the union of the abilities of the already considered attackers.

- Nation state without regard for international law and convention (e.g.: USA)
  - Pressure those it claims governance over into doing as it demands
  - Pressure companies operating within it into colluding in an attack
  - Identify all people connecting to the server. (Formed from the union of powers of the ISP and the server owner and operators)
- ISP (e.g.: BSkyB)
  - View all traffic on their network, after the point at which a user comes under suspicion.
  - Manipulate all traffic on their network however they desire.
  - Identify an IP address (during a specific time) with a person.
- Server Owners and Operators (i.e.: Those who own and operate Turtlenet)
  - Alter the source of the server in any way they desire.
  - Log all traffic before and after a user comes under attack.
  - Manipulate all traffic in any way they desire.

- Collect the IP of all connecting users.

Some of these claims may seem extreme, but given that companies such as BT, Vodafone Cable, Verizon, Level 3, and others have provided unlimited access to their networks[**tempora**] to governmental spy agencies, we feel it is a reasonable threat model in light of recent revelations[**wikiDisclosures**].

Given that our system is intended to both protect people from the governments which claim governance over them, and mere greedy companies looking to sell or collect user data for profit, we will assume the worst case: i.e. that all our users, their ISPs, and the owners and operators of the Turtlenet server they use are able to be pressured by the adversary.

We grant the adversary all the powers listed above, and assume that all ISPs, companies, and Turtlenet server operators are actively working against all of our users. In summary, we consider the adversary to be:

> A nation state for which money is no object, claims governance over the user, and has the ability to pressure service providers into spying on their users.

4.1 Scope

We do not attempt to protect against an adversary who has access to and the ability to modify the users hardware, nor do we attempt to conceal that an IP uploads data to the network.

While we recognise that the ability to post messages anonymously is important, especially considering that countries normally considered benign are prosecuting people over whom they claim governance for saying 'offensive' things [**illegalOpinions**], it is unfortunately out of scope for this project.

Chapter 5

Anticipated Software

We anticipate the creation of the following software:

- Windows, Linux, and OSX executable: client
- Windows, Linux, and OSX executable: server
- Windows, Linux, and OSX executable: installer for client and server
- Full source for server, client, and any associated works

The client will create and use an SQLite database, local to each client, this database will be used to store all information that the specific client is aware of.

Chapter 6

Anticipated Documentation

We will provide the following documentation:

- Installation guide for a server
- User manual for a client
- Full protocol documentation for third parties wishing to implement their own clients
- Full description of system design and architecture, for future maintenance
- Full description of database design
- Interface documentation

Chapter 7

Anticipated Experiments and Their Evaluation

7.1 Performance Testing

Evaluating how well the system performs under a high work load.

- Test to see how many simultaneous clients the server can handle.
- Test to see if the data received from the server under a high work load is accurate.
- Test the impact of a large number of clients on the servers response time.

A high work load will be simulated by automated clients performing user actions at random. The server should be capable of allowing these clients to communicate with one another quickly. The maximum number of concurrent clients possible without noticeable lag (twice the frequency of updates) should be recorded.

7.2 Robustness Testing

System level black box testing.

- Devise a series of inputs and expected outputs.
- Run these inputs through the system and record the actual outputs.
- Compare the actual outputs with the expected outputs.

- Simulate a denial of service attack. The server should be able to recover from the attack quickly and with minimal impact on the clients. Blocking such an attack is beyond the scope of this project.

Inputs used should range from expected use patterns to silly as users tend to do things totally unexpected. Expected and actual outputs should be recorded. Any differences will indicate problems with the system which need to be fixed.

7.3 Recoverability Testing

Evaluating how well the application recovers from crashes and errors.

- Restart the computer while the application is running. Ensure the local database is not corrupted.
- While the application is running terminate the computers network connection. Ensure the application continues working after the connection is re-established.
- Send a badly formatted message to another client. Ensure the application is able to keep running after receiving unexpected data from another client.

Each test should be run several times. If any test fails once or more this indicates that the system is bad at recovering from crashes and/or failures. In the case of a failure changes to the system should be implemented to improve recoverability.

7.4 Learnability Testing

Trialling the user interface with non expert users. Users should be able to use the system with minimal frustration and, ideally, without consulting the manual.

- Ensure users understand how to add friends, send messages, create posts, comment on posts and like posts.
- Ensure users don't spend excessive time searching for functions within the interface.
- Ensure error messages can be understood by the user and offer understandable advice on how to proceed.

Each test should be run several times with different users. If more than one user fails a test then changes need to be made to the interface. A single user experiencing problems is not an indication of a problem with the interface but instead suggests user incompetence.

7.5 Security Testing

The main goal of the system is to be secure. To ensure this goal is met the security of the system should be tested.

- Send non standard messages to clients. These should be rejected. If there is a flaw in the system the client may reveal information unintended for the recipient, in this case the program sending non standard messages.
- Recruit experienced programmers from outside of the group to attempt to penetrate or otherwise break the system. All attempts should be unsuccessful.

If any test fails this indicates a vulnerability in the system which should to be corrected immediately. Security tests should be rerun after any changes during the testing phase to ensure new vulnerabilities are not introduced.

Chapter 8

User View

The user will be presented with a simple and easy to use interface, which assumes and requires no knowledge of security. The most complicated thing that the user will have to do is transmit to other users their public key. We plan to alleviate this process by encoding the public key as both a QR code and plaintext string (depending on user preference), both of which may be easily transmitted via email, SMS, meeting in person, or over any other channel.

Upon connecting to the system for the first time, the user will be prompted to enter a username, and any profile information they choose to share, and a passphrase. They will be urged to avoid using their real name as their username, and informed that profile information is shared on a case by case basis, and is not automatically visible to people whom they add. The entered passphrase will be used to deterministically derive an AES key which will be used to encrypt the users keypair and local DB. The user will be given the option of creating a second passphrase which, when entered, will overwrite the keypair and local DB with random data.

They will then be brought to the main page of the system, where they (and) people they authorize, may post message. There will be a prompt for them to add peoples public keys, and the option to add either a QR encoded or plaintext encoded public key.

Upon adding another's public key, they will first be informed of that persons username, and prompted to categorise the person. The user will be able to create a number of categories into which they can place that user. Already created categories will be displayed. One person may be added to multiple categories, and nobody but the user is aware that this occurs. Depending upon the categories the person is entered into, that person gains the ability to view certain content posted by the user.

When the user posts a message they are prompted to enter a recipient, this may be: a previously created category (such as friend, co-worker); a number of individuals; or any combination thereof.

Upon receiving a message a sound is played and the user is informed. They are then able to

click on the notification to open the message, and chat. When chatting with another user they have the ability to 'ignore' that user, in this case the user will see no more messages from that user.

8.1 System Boundary Diagram

Each client (of which there may be many) has his own client boundary consisting of his database and program client, whilst the server operators have their own boundary consisting of just the server. You can see in the diagram that at no point does the server operator or user functionality coincide with each other, leaving their privacy fully independent of one another. Each client (of which there may be many) has his own client boundary consisting of his database and program client, whilst the server operators have their own boundary consisting of just the server.

We can see the users interaction with the system below in the System Boundary Diagram:

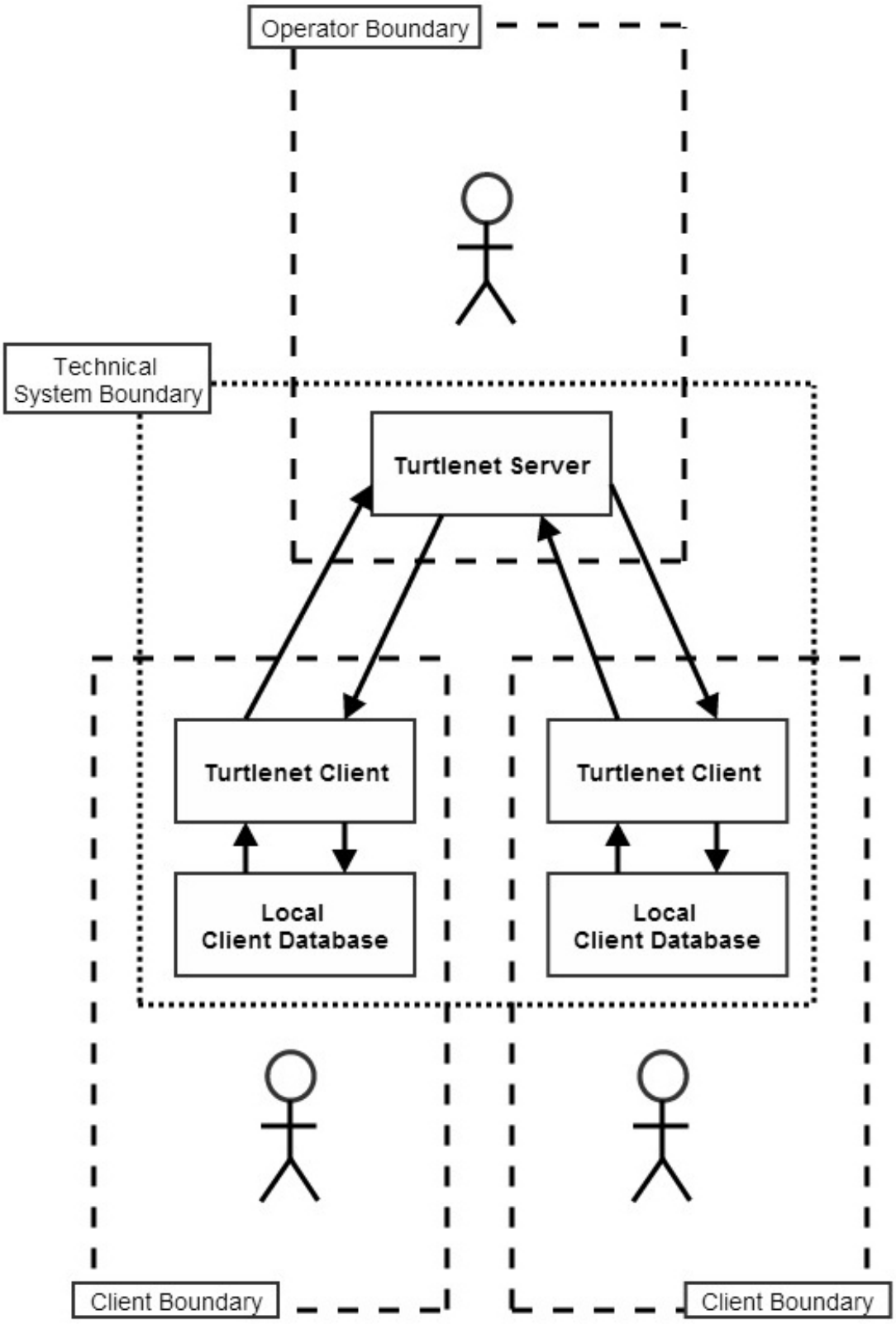


Figure 8.1: System Boundary Diagram

Chapter 9

User Requirements

9.1 Registration

Users may register by sending a CLAIM message to the server, this will claim a username for that user, and allow people they send messages to to see their username.

Before registering the user must generate an RSA keypair, they will be given the option of generating a new keypair, or using an existing keypair. The keypair provided will be encrypted using AES with the users password being used to derive the key. The user therefore must enter their password to log into the client. The database will be encrypted using the same AES key as the keys are encrypted with.

9.2 Interacting with other users

People are added by adding their public key, this is transmitted outside of our system, via whichever channel the users deem appropriate[1]. We will provide a user with their public key as a QR code, or a plaintext string, depending on user preference.

Adding someone is asymmetric. Just because you add them doesn't mean they've added you. You do not require consent to add someone, just their public key.

The system allows the user to manage their list of known people into categories such as friends, family, and co-workers. The user defines these groups as lists of people whose public key they know. The user may create any group they desire, these groups are visible to only the user, and private.

[1] This is required to prevent server operators from MitM'ing users

9.3 Profile Data

Profile data will be transmitted via PDATA messages. Different versions of profile information may be provided to different groups of people. Profile data may be updated by the user by future PDATA messages.

The supported fields in a PDATA message are:

- Name
- Username (unique, but this uniqueness is ensured by server and shouldn't be relied on)
- Birthday
- Sex
- E-Mail
- About
- Misc.

9.4 Account recovery

Account recovery is not possible without your keypair, due to this the the GUI should urge the user to keep a copy on a flash drive, or external hard drive. The keys themselves will be encrypted with the users password.

9.5 Posts

9.5.1 Walls

Each user has their own wall. On their wall they may posts messages for themselves and others to see. All wall posts should be addressed to the user themselves so they can see their own posts, otherwise they will be unable to even view their own posts. When posting to their wall they choose who will be able to see the post, whether this is a group or people, a specific list of people, or just themselves is up to the user. They will not however be given the option to post publicly. Users may also post to another users wall.

Wall posts may contain links to other content, however this content is never thumb-nailed[2].

[2]client MUST NEVER thumbnail links or otherwise access external content without EXPLICIT user consent (see tor/js exploit on freedom hosting by the USA and tracking techniques recently thwarted by GMail now caching images. Specifically the fact that by delivering content over a secure channel that initiates communications outside of that channel, the recipients of content may be identified. A common variation of this is 'pixels' whereby a

A user may edit their old posts, however older versions will still be available for viewing; similarly users may 'delete' posts, but they are still visible to malicious clients.

Due to bandwidth limitations on such networks as we are building, a user may only post plaintext, they may not post images, video, or audio.

9.5.2 Commenting and Liking

All wall posts may be commented on by any user who can see them. Comments are visible to all people who can see the original post; due to this, comments must be forwarded by the original posters client to all the same recipients, as the commenter may not know whom the original posters allowed to see the post.

Any wall post, comment, or other item on a wall may be liked.

9.5.3 Events

The client will alert the user to other users birthdays by automatically posting a wall post that only the user may read, which alerts the user of the event. These are otherwise normal wall posts. The user has the option of setting a category of people as a group for whom they desire to be alerted of events regarding.

Furthermore users may create their own events, for themselves and others to be alerted to. Recipients of events they did not create must accept the event before they are alerted of it.

9.6 Chat

Users may chat in real time, however messages can still be sent when one user logs off, to be received when they log in. Past conversations are saved, and a user may block users from messaging them; the client actually just ignores their messages, it's impossible to stop someone from messaging you.

Conversations may include two or more people.

---

would be tracker embeds a 1x1 transparent PNG in a document, and watches who downloads that image from their servers.[**webbug**]

Chapter 10

Case Study: Facebook

10.1 Overview

A user has a profile with information about them, they may add other users as 'friends', friends may view each others 'posts' and talk to each other. Posts are multimedia messages typically visible to all the friends of the person who made the post. Most posts can be commented upon, and both posts and comments may be 'liked'. Liking merely publicly marks the fact that you approve of something.

10.2 Registration

In order to be a user of facebook, one must register. In doing so you provide facebook with the following information, this may also be used to later reset the password of your account, should you forget it.

- First Name
- Last Name
- E-Mail
- Password
- Birthday
- Sex

In order to register one must read and agree to their terms [**fbterms**], read their data use policy [**fbdatause**], and read their cookie policy [**fbcookies**]. Given profile information can be changed at a later date, within certain bounds. Facebook requires the use of your real name, and in fact forbids all false personal information, under their terms.[**fbterms**]

10.3 Account Management

The user is given the ability to set the security defaults for their posts and information. These options include who is able to see wall posts, whether comments are enabled by default, and who may see which aspects of your profile information. You can also manage the permissions granted to facebook apps.

Access may be gained to an account by knowing certain information, the intent is to allow people to recover their account if they forget their password.

A users profile may contain the following information:

- Work and education
- Place of Birth
- Relationship
- Basic Information
  - Birthday
  - Relationship
  - Status
  - Anniversary
  - Languages
  - Religious
  - Political
  - Family
  - Contact Information

10.4 Friend

In facebook, 'friending' someone is symmetric; that is, if you are friends with them, they are friends with you. The facebook severs store which user is friends with which other users. Adding another

Field	Description
Photo	All the photos the user's has tagged
Friend	What friends the user has
Note	What notes the users up/downloaded to facebook
Groups	What groups the user has join
Events	What events user may be attending
Likes	What page(s) (unknown type) the user liked
Apps	What apps the user has
Books	What book pages the user liked/followed
TV programmes	What TV pages the user liked/followed
Films	What films pages the user liked/followed
Music	What music(or stars) the user liked/followed
Sports	What sport pages the user liked
Place	Where's the user has been

Table 10.1: The user adds a new post

user as a friend is simply a matter of sending that user a friend request, and having it approved by the second user. A user may see a list of all who are their 'friend' on FB, in the friend list. After friending somebody that persons wall posts will appear on your news feed, and you will be able to chat with that user.

In order to add friends, facebook allows you to see your friends friend lists, and search by name, email, and location for other users. Facebook also suggests other users whom you may already know IRL, based on your friends friends. Non-users are also able to search facebook for people that they may know.

10.5 Post

10.5.1 Posts, and functions thereof

Facebook allows a user to post on their wall or friend's wall (if they are friens with the facebook user). Posts may contain: text, images, videos, or any combination thereof.

A user posting a post may do the following:

- Delete their own post
- Rewrite their own post
- Decide who may view a post, the options are as follows:
  - Public
  - Private

- Only-me
- Friends only
- Friends of friends

10.5.2 Interaction with another's posts

A post will typically be displayed on the news feeds of the people who are able to see it, due to this the name of the person who made a post is always displayed next to it. Posts themselves may be commented upon, liked, and reposted to the viewers wall ('shared') with an additional message; the number and names of people who have liked a post is displayed underneath it; likes may be cancelled at a later date. The comment function however, may be disabled by the user who makes the post.

A user may hide specific posts, or hide all posts by a specific user. They may also, instead of hiding another's posts all together, merely prevent them from being automatically displayed on their news feed. A user may report an image, video or comment to facebook team (e.g. the post is offensive). Comments may also be liked, hidden, and reported; following such a report FB is able to remove offensive or illegal posts.

Images which are posted may be tagged, this allows other users to mouse-over parts of the image and be informed who is pictured. This functionality is also used to add all posted images of someone to their profile.

10.6 Wall

A users wall stores all the posts of the user posted since the account was created and the information about the user, this information is presented in reverse chronological order, so that recent events are at the top of the page and easily visible. Other users may view the users wall by clicking the name of the user from anywhere in facebook. Other users may post on a friends wall along with it's owner (see section on posts for more information); in this case, both the poster and the owner of the wall can delete the post. Facebook also retains the power to erase any content on its service.

Posts mentioning a user are automatically reposted to that users wall, this can occur manually or when that person is tagged in an image.

10.7 Chat System

Facebook allows a user to chat with their friends, and will inform a user of whether their friends are online or not (though this can be faked), and whether the user you are chatting with has read

the last message that you sent them. You are also informed whether your friend is logged in on a mobile device or not.

Whole groups of users may chat together, in multi-user conversations. Facebook also supports video calling and file transfer during chats. If a user does not wish to be bothered by another using chatting with them, then they may 'mute' that users conversion. Users spamming via chat may be reported to facebook. Because multi-user conversations (and indeed long running one-to-one conversations) can get rather large, facebook allows you to hide the history of a conversation.

Facebook chat alerts the user to new messages in a conversation by playing a sound.

10.8 Architecture

From a users point of view facebook is ostensibly organised as a single central server; we are here concerned with the general architecture and not the specific implementation of it, and so we will consider all of facebooks servers to be a single server for the purposes of this section.

Users connect to facebook using a web browser, and proceed to download a client written in javascript. User data is uploaded to facebook over HTTP as cleartext. The data is stored on unencrypted on facebooks servers, and facebook maintains a database of all data.

This allows clients to download only the data they need, as they can simply ask for it. This in turn means that facebooks current architecture can, and does, support a huge user-base, measured in the millions.

10.9 Security

In order to use facebook after registration a user must 'log in'. This places an authentication cookie on the users computer which gives anyone in possession of it the ability to act as that user.

If the user logs in from an IP associated with a region geographically far from the last login, facebook will confirm that the user owns the account by asking them to identify a friend in a photograph, or by other means.

Facebook chat turns the users computer into a server, whereby facebooks central server sends messages to the client as it receives them, rather than the client requesting new messages. This has been used in the past to identify facebook users by correlating sent messages of specific size sent at a specific time.

Facebook has access to all its users data, and is able to erase, modify, and fabricate it. Facebook is aware of everything which happens on facebook. Censorship is a common occurrence on facebook.

Chapter 11

Case Study: Tor

11.1 Overview of Protocol

Tor is an implementation of onion routing, it routes traffic from your computer through a number of other nodes; the final 'exit' node the routes the traffic to the final destination[**torProtocol**]. Node IP's are listed publicly in directory servers. In this manner the IP of clients connecting to a server is obscured from that server.

RSA/AES is used to ensure that only you, the exit node, and the final destination see the plaintext traffic being routed. With the use of TLS, SSL, or other end-to-end encryption those who see the plaintext can be reduced to you and the final destination. However a malicious exit node can MitM SSL connections using ssl-strip or a similar tool. There are methods of avoiding this, but it is a serious issue because users believe that SSL is secure. This exploit is found in the wild[**badRelays**], and so is most definitely a concern.

Tor also supports 'hidden services' which seek to conceal the IP of the client from the server, and the IP of the server from the client. These are significantly more secure as the traffic never exits the tor network, however provide no protection from the adversary as will be described later; after all, we're assuming the server operators are colluding, so they will provide data required for traffic confirmation.

11.2 Security

Given that Tor is a low-latency network, traffic can easily be correlated. This problem is ameliorated in high-latency networks such as mix nets, but not eliminated.

Tor does not seek to protect against size correlation, or time correlation of traffic. Rather the purview of tor is to conceal the IP address of a client from the servers which it connects to.

Should a global passive adversary have perfect visibility of the internet, they would be able to track tor traffic from source to host by correlating the size and time of transmissions.

> The Tor design doesn't try to protect against an attacker who can see or measure both traffic going into the Tor network and also traffic coming out of the Tor network[**torOneCell**].
> - Roger Dingledine

We can safely assume that the adversary has access to the clients traffic, since our threat model is that of a nation state seeking to spy on its citizens. Furthermore we may assume that the adversary has access to the content host, as our threat model assumes that service operators may be pressured legally or otherwise into spying on their users. Therefore we must conclude, at least for *the* adversary, that Tor is unsuitable for concealing activity in traditional social networks, due to traffic confirmation.

Does this then mean that Tor is insecure? No. So far as we know[**torStinks**] the US does not currently have the ability to reliably and consistently track tor users; if the US is incapable of doing so, it is reasonable to assume that no other nation state has this ability. This is however not something which should be relied upon, as assumptions widely lead to mistakes. We shall therefore consider Tor as unsuitable for transmitting our data, at least if we were to do so as a traditional social network.

> With manual analysis we can de-anonymize a very small fraction of Tor users, however, no success de-anonymizing a user in response to a TOPI request/on demand[**torStinks**].

Chapter 12

Case Study: GPG and Email

GPG is an implementation of the PGP[**rfc4880**], providing both public/private key encryption and also a number of symmetric ciphers that can be used separately.

It is common practice to use GPG to encrypt email, and several popular addons for browsers exist to aid in this[**gpgaddon**]. Unfortunately GPG itself is difficult to use[**greenwaldAnnoying**], and a significant barrier to entry.

The encrypting of email with RSA[1] is a good solution if one wants to keep the content of messages secure, and unmodified. However it is out of scope for PGP to hide who is communicating, so while we find the underlying cryptography sound, our scope is simply too different for PGP to be of any use; with one exception.

Public key distribution is a significant challenge [2]. PGP partially solves this problem by introducing the concept of a 'web of trust'. In such a system one marks public keys as trusted, presumably the keys of people you trust, and the people whom you have marked as trusted can then sign the keys of other people whom they trust. These keys may then be distributed, with the RSA signatures of everyone who signed them, to everyone. If I download a key and see that it has been verified by someone that I trust, then I can trust that key (albeit less than the original key). This in combination with the small word hypothesis[3][**sixdegrees**] allows a large number of public keys to become known to a user merely by adding one friends key, and having the client automatically sign all keys it comes across from a trusted source.

We will take the 'web of trust' into consideration during design, however it may present some significant security issues.

[1] and a symmetric cipher

[2] Our system can't do it, or it would be trivial to MitM users who don't check they received the correct key via another channel

[3] The phenomenon that people in the earth's population seem to be separated by at most 6 intermediaries.

Chapter 13

Case Study: alt.anonymous.messages and Mix Networks

alt.anonymous.messages is a newsgroup[1] to which people publicly post encrypted messages. In order to retrieve messages a recipient downloads all new messages and attempts to decrypt them all, those which they are able to decrypt are read, and others ignored.

This type of system is known as a 'shared mailbox', and is often not used by hand, but by mix network servers, which provide high-latency email forwarding, and handle the encryption on behalf of the users. Mix-networks massively slow timing-based traffic confirmation because they cache a large number of messages before sending them all out at once in a random order[**mixnetOperation**].

This system provides the property we are seeking: concealing who talks with whom on our network, even from the server itself. This property is ensured by the fact that the server cannot tell who reads a specific message, even though it knows which IP uploaded it. It also introduces a huge amount of overhead, in the form of downloading everyone else's messages as well as ones own.

Mix networks however have some serious issues, and misconfiguration easily allows for traffic correlation[**mixnet**], albeit not confirmation (without a large sample size). Furthermore mix networks only function if the operator is trusted, this is unacceptable against our threat model. For these reasons we will not use the idea of mix networks.

We have identified the method of operation of shared mailboxes as the basis for our communications protocol, and will build a social network on top of this concept.

[1] It may be accessed, without an installed Usenet client, through several websites providing an interface to it, one such website is Google Groups, and may be accessed here: https://groups.google.com/forum/#!forum/alt.anonymous.messages

Chapter 14

Security and Usability

14.1 Security and Usability

There is room for improvement when it comes to increasing the level of usability and user-friendliness in the Turtlenet software.

Firstly, as developers, we need to pay attention to the target audience, especially when it alludes to the general public, which have different ranges of user experience and knowledge when using a social network program. Paying attention and understand to how the general public thinks and the level of their knowledge are, we as developers can shape the program into something that is user-friendly and avoid mistakes which can be easily made by the users if the program is confusing. We have to be aware that since Turtlenet is a social media which is more security oriented, there are some functionalities that the general public might not be knowledgeable about.

In order to help the users who uses Turtlenet for the first time in our future development, is the use of 'pop up manual'. These instructions will pop out in a subtly manner in specific areas on the website when the user uses Turtlenet for the first time. An example is under the public key field. The general public would have no idea what a public key is or what is to do with it. A pop up manual is useful for this case, it will pop out saying in a generic, non-jargon or technical sense saying "Public key is the key to share with your trusted friends! Hence its name, it behaves like a key which can be passed along to others so they can unlock and view your profile."

Lastly, in order to make future improvements of the program, one important and effective way to do so is to listen to the customers' feedback. There are several platforms to gather feedback from customers, one common way to do it nowadays is using social media websites, and Twitter will be the best option for this case. Users can easily mention us in tweets, could be anything from actual feedback, rants, questions regarding the use of the program, what they like about it, and positive feedback. This could become useful to us developers to see how the customer think, feel and behave

towards Turtlenet, and possibly the trends about their use of it. Using such information, developers can improve on certain areas of the program according to the users needs. Users are the one who uses the program regularly and know what is best when it comes to functionality.

Chapter 15

System Requirements

An estimate is hereafter given as to the size of all stored messages, and the amount of data which would need downloading by each client when it is started. The following assumptions are used throughout:

- A users average message posted to their wall is 200 characters
- A users average number of messages posted to their wall per day is 10
- A users average number of friends is 100 (each and every friend represents one key exchange)
- A users average private message (to single user) is 50 characters
- A users average number of private (to single user) messages per day is 300

With these generous estimates, each user would generate (200*10*100)+(50*300*1) bytes of raw data per day. Assuming a 10% protocol overhead we would see 236,500 bytes of data per day per user.

The storage space required for a server is therefore 86MB per year per user. On a server with 50,000 users that has been running for 3 years, there would be just 1.3TB of data.

Every time a client connects, it must download all messages posted since it last connected to the server. To mitigate this we may run as a daemon on linux, or a background process in windows, that starts when the user logs in. If we can expect a computer to be turned on for just 4 hours a day then 20 hours of data must be downloaded. ((236,500*no_of_users)/24)*hours_off_per_day bytes must be downloaded when the users computer is turned on.

The following table shows the delays between the computer turning on, and every message having been downloaded (assuming a download speed of 500KB/second, and a network of 1000 users).

Hours off per day	Minutes to sync
0	0
4	1.3
10	3.2
12	3.9
16	5.2
20	6.5

Table 15.1: Hours a computer is turned off per day vs minutes to sync

We feel that waiting 2-5 minutes is an acceptable delay for the degree of privacy provided. Once the user is synced after turning their computer on, no further delays will be incurred until the computer is shut down.

Due to the inherently limited network size (<1500 users of one server is practical) we recommend a number of smaller servers, each serving either a geographic location, or a specific interest group.

While this latency could be avoided, and huge networks (>1,000,000) used, it would come at the cost of the server operator being able to learn that somebody is sending or receiving messages, and also who those messages are sent to/from (although they couldn't know what the messages said).

The server therefore merely needs a fast internet connection to upload and download content from clients. The client is required to perform a significant amount of encryption and decryption, however the client will almost certainly be able to encrypt/decrypt faster than a connection to the internet so the network speed may be considered the limiting factor for users on the internet [**cryptoSpeed**]. Large companies however may very well use the system over a LAN, however these can be reasonably expected to have fairly modern computers which can more than handle RSA decryption.

Chapter 16

Transaction Requirements

Due to the nature of Turtlenet there may exist no central database, rather each client maintains their own local database of everything they know. The data forming this is all stored centrally, however to build a complete database would require the private key of every user of the service, which clearly we do not have access to.

There are 3 categories of data transaction:

- 1. Data entry
- 2. Data update and deletion
- 3. Data queries

16.1 Profile creation of the user

All that is required to use a Turtlenet server is a valid RSA keypair. Users don't have accounts per se, but rather associate profile data with a public key if they so desire. Users have no login information, rather posts are authenticated via RSA signatures. Usernames are the sole public information in our system, and as such each client has a complete list of usernames.

When a client first connects it is advisable, albeit not required, to claim a username. This is done merely by posting that username, and a signed hash of it to the server. Therefore the DB must store all such CLAIM messages.

Optional profile data which the user may enter is stored as PDATA messages, and the databse will be required to store these.

16.2 Adding of user relations

Communication between people on Turtlenet requires that one is in possesion of the public key of the recipient, and should they wish to respond then they must be in possesion of your public key. We define 'A being related to B' to mean that A is in possesion of B's public key, and B is in possesion of A's public key. This is given a special name as it is a very common situation.

A user may be uniquely identified by their public key, and it may be used to derive their username, if they claimed one. Being in a relation with someone doesn't mean that you can see any profile information of theirs, however the GUI will ask the user whether they wish to share their own profile information with someone when they add that persons key.

16.3 Assigning relations into categories

When a user adds a relation, he has a choice of adding him into a specific category (or categories). A user can create any category he wants by going to the options and click 'Add new category'. The database then records the new category into the category table. The user then can then assign the relation into the existing category.

Categories are useful because they allow the user to share their posts with a predetermined set of people automatically, withing having to list each individual as a recipient.

16.4 Adding of posts

Users may post on their, or - with permission - others, walls. A post has a list of people who can see it (the user may choose a previously defined category or a specific list of people) however this list isn't public so only the DB of the author of a post (and the owner of the wall) will contain information as to who is able to see it.

The post itself has a timestamp, a signature (authenticating it), and content. The databse will store all of these.

NB: When a user posts something, they are automatically added to the list of recipients. A users own posts are downloaded from the server, just like everyone elses, and are in no way special.

16.5 Adding of events

The database will store events, these may be created by the user, or recieved from other users. At the appropriate time the GUI will notify the user of an event occuring. Example include birthdays, deadlines, and important dates. Events recieved from other users must be accepted by the user

before the GUI will alert the user of them, for this reason the DB must also store whether an event is accepted or not.

16.6 User creating a new message

A user can initiate a conversation with (an)other user(s) by creating a new message. Messages are merely a special case of wall posts, which are handled differently by the GUI.

16.7 Receiving Content

When the client connects it will download all messages posted since it last connected, it will then attempt to decrypt them all using the users private key. Those messages which are successfully decrypted are authenticated by verfying the signature and the content added to the database. It is in this manner that all content is passed from server to client.

Chapter 17

Task List

Task ID	Task Description (Desc.)	Due Date	Deliverable
1	Project Planning	14/02/2014	Planning segment
1.1	Mission Statement	07/02/2014	Same as Desc.
1.2	Mission Objectives	07/02/2014	Project Goals
1.3	Project Target	07/02/2014	Project Scope
1.4	Threat Model	07/02/2014	Project Scope
1.5	System Requirements	07/02/2014	Same as Desc.
1.6	User View and Requirements	07/02/2014	Same as Desc.
1.7	Transaction Requirements	07/02/2014	Same as Desc.
1.8	Case Studies (CS)	14/07/2014	Eval. of rival
1.8.1	CS: Facebook	14/07/2014	Eval. of rival
1.8.2	CS: GPG and E-Mail	14/02/2014	Eval. of rival
1.8.3	CS: Tor	14/02/2014	Eval. of rival
1.8.4	CS: 'aam' and mix networks	14/02/2014	Eval. of rival
1.10	Risk Assessment	14/02/2014	Same as Desc.
1.11	Anticipated Software	14/02/2014	Project Estimates
1.12	Anticipated Experiments and Evaluation	14/02/2014	Project Estimates
1.13	Anticipated Documentation	14/02/2014	Project Estimates
1.15	User View	14/02/2014	Same as Desc.
1.16	Gantt Chart	14/02/2014	Same as Desc.

Task ID	Task Description (Desc.)	Due Date	Deliverable
2	Project Design	14/03/2014	Design Segment
2.1	Research (Res.)	21/02/2014	Research Segment
2.1.1	Res: Database Languages	21/02/2014	Same as Desc.
2.1.2	Res: Programming Languages	21/02/2014	Same as Desc.
2.1.3	Res: Interfaces	21/02/2014	Same as Desc.
2.2	Designs (Des.)	07/03/2014	Design Segment
2.2.1	Des: Databases	28/02/2014	Same as Desc.
2.2.2	Des: Class Interfaces	28/02/2014	Same as Desc.
2.2.3	Des: Protocol	28/02/2014	Same as Desc.
2.2.4	Des: Architecture	28/02/2014	Same as Desc.
2.2.5	Des: Sequence Diagrams	28/02/2014	Same as Desc.
2.2.6	Des: Data Flow Diagrams	28/02/2014	Same as Desc.
2.2.7	Des: Class Diagrams	28/02/2014	Same as Desc.
2.2.8	Des: Server-side Interfaces	28/02/2014	Same as Desc.
2.2.9	Des: Client-side Interfaces	28/02/2014	Same as Desc.
2.2.10	Des: Server-side Protocols	28/02/2014	Same as Desc.
2.2.11	Des: Client-side Protocols	28/02/2014	Same as Desc.
2.2.12	Des: Server-side Pseudo-code	07/03/2014	Same as Desc.
2.2.13	Des: Client-side Pseudo-code	07/03/2014	Same as Desc.
2.3	Segment Review	10/03/2014	Design Segment
2.3.1	Evaluate Segment Quality	14/03/2014	N/A
2.3.2	Improve Segment	14/03/2014	Design Segment

Task ID	Task Description (Desc.)	Due Date	Deliverable
3	Implementation stage (Imp.)	28/04/2014	Imp. Segment
3.1.1	Imp: Architecture	21/03/2014	Work Environment
3.1.2	Imp: Architecture Docs	21/03/2014	Documentation
3.2.1	Imp: Target System (TS)	21/03/2014	Work Environment
3.2.2	Imp: TS Documentation	21/03/2014	Documentation
3.3.1	Imp: Databases	21/03/2014	Database
3.3.2	Imp: Database Documentation	21/03/2014	Documentation
3.4.1	Imp: Server-side Protocols	28/03/2014	Program function
3.4.2	Imp: Server Protocol Docs	28/03/2014	Documentation
3.5.1	Imp: Client-side Protocols	28/03/2014	Program function
3.5.2	Imp: Client Protocol Docs	28/03/2014	Documentation
3.6.1	Imp: Server-side Interface	04/04/2014	Interface
3.6.2	Imp: Server Interface Docs	04/04/2014	Documentation
3.7.1	Imp: Client-side Interface	04/04/2014	Interface
3.7.2	Imp: Client Interface Docs	04/04/2014	Documentation
3.8.1	Imp: Server-side Source Code	18/04/2014	Program
3.8.2	Imp: Client-side Source Code	18/04/2014	Program
3.9.1	Imp: Server Install Docs	18/04/2014	Documentation
3.9.2	Imp: Client Install Docs	18/04/2014	Documentation
3.10	Segment Review	28/04/2014	Imp. Segment
3.10.1	Evaluate Segment Quality	28/04/2014	N/A
3.10.2	Improve Segment	28/04/2014	Imp. Segment

Task ID	Task Description (Desc.)	Due Date	Deliverable
4	Project Portfolio	09/05/2014	Portfolio
4.1	Fabricate Reports	02/05/2014	Reports

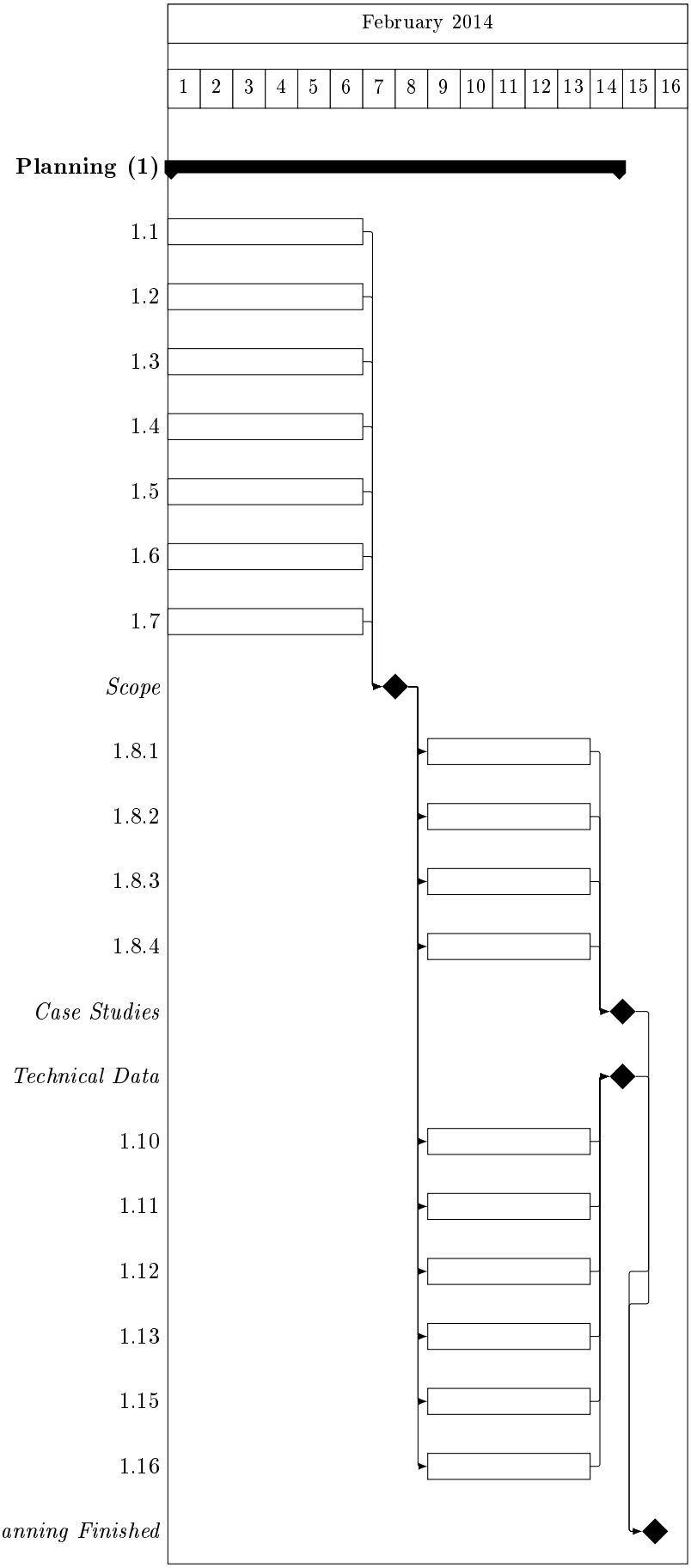
Chapter 18

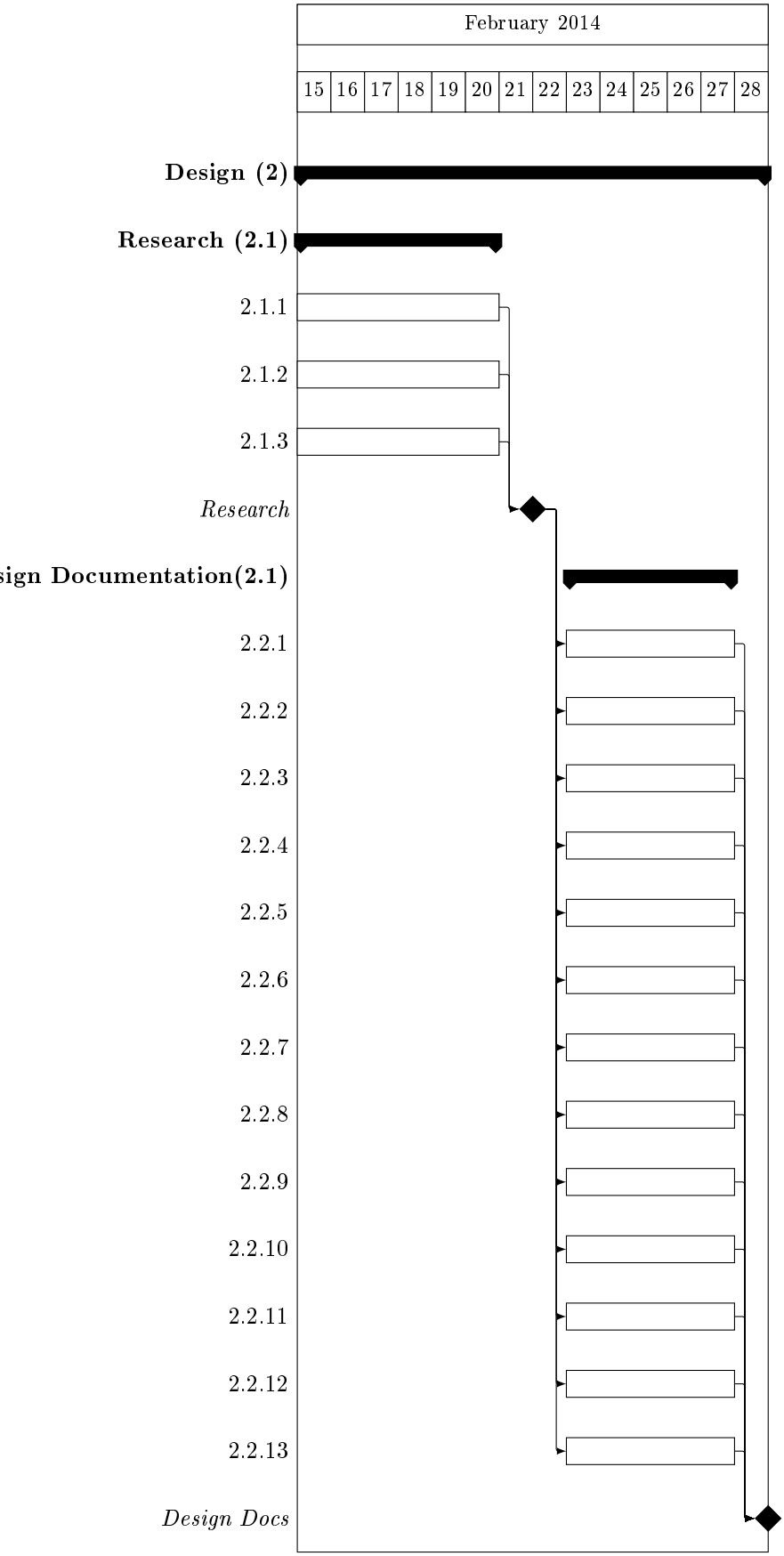
Gantt Chart

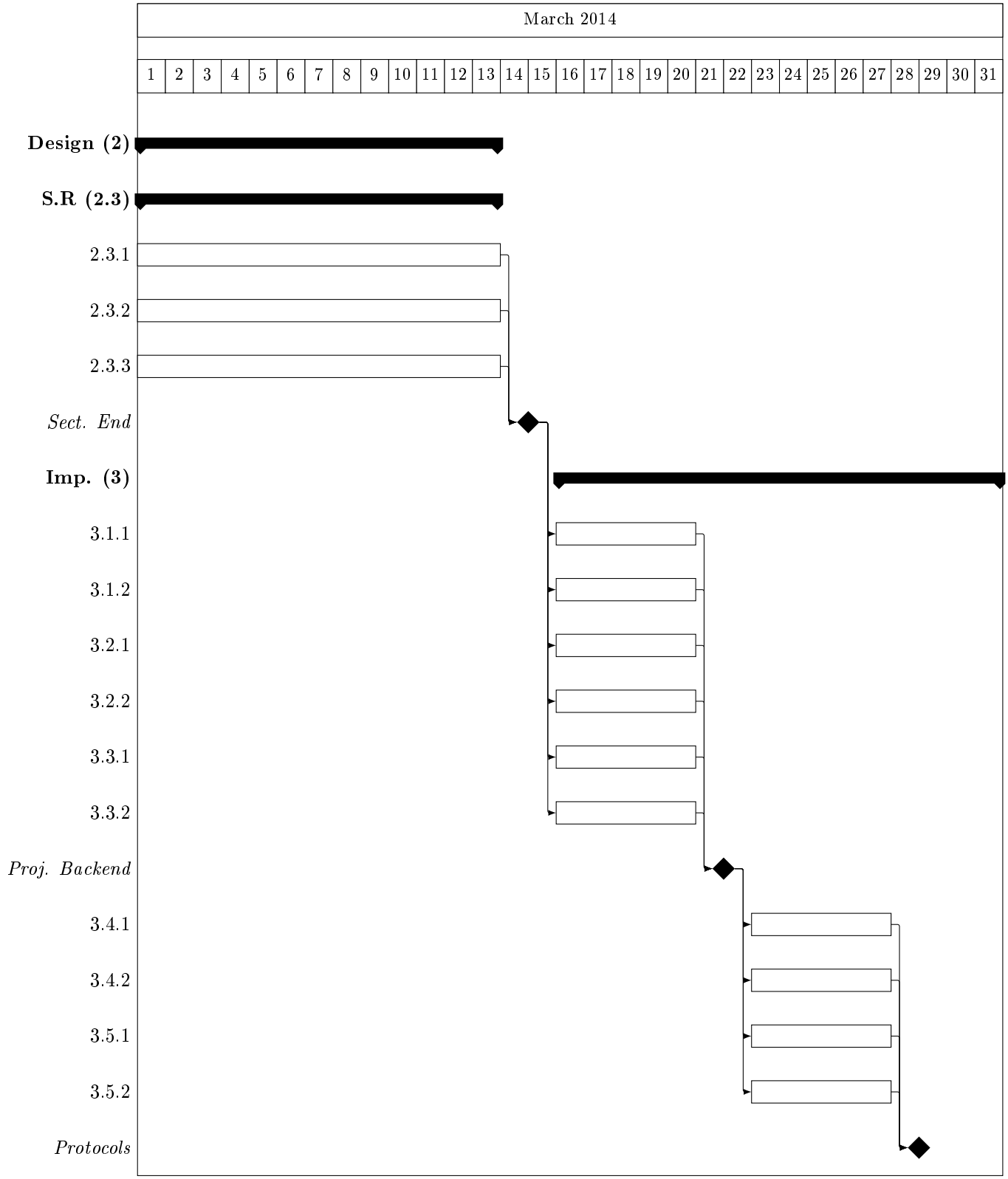
The following are Gantt Charts of the project. They are developed from looking at overall requirements for the project, and to act as a base for us to follow when developing the project.

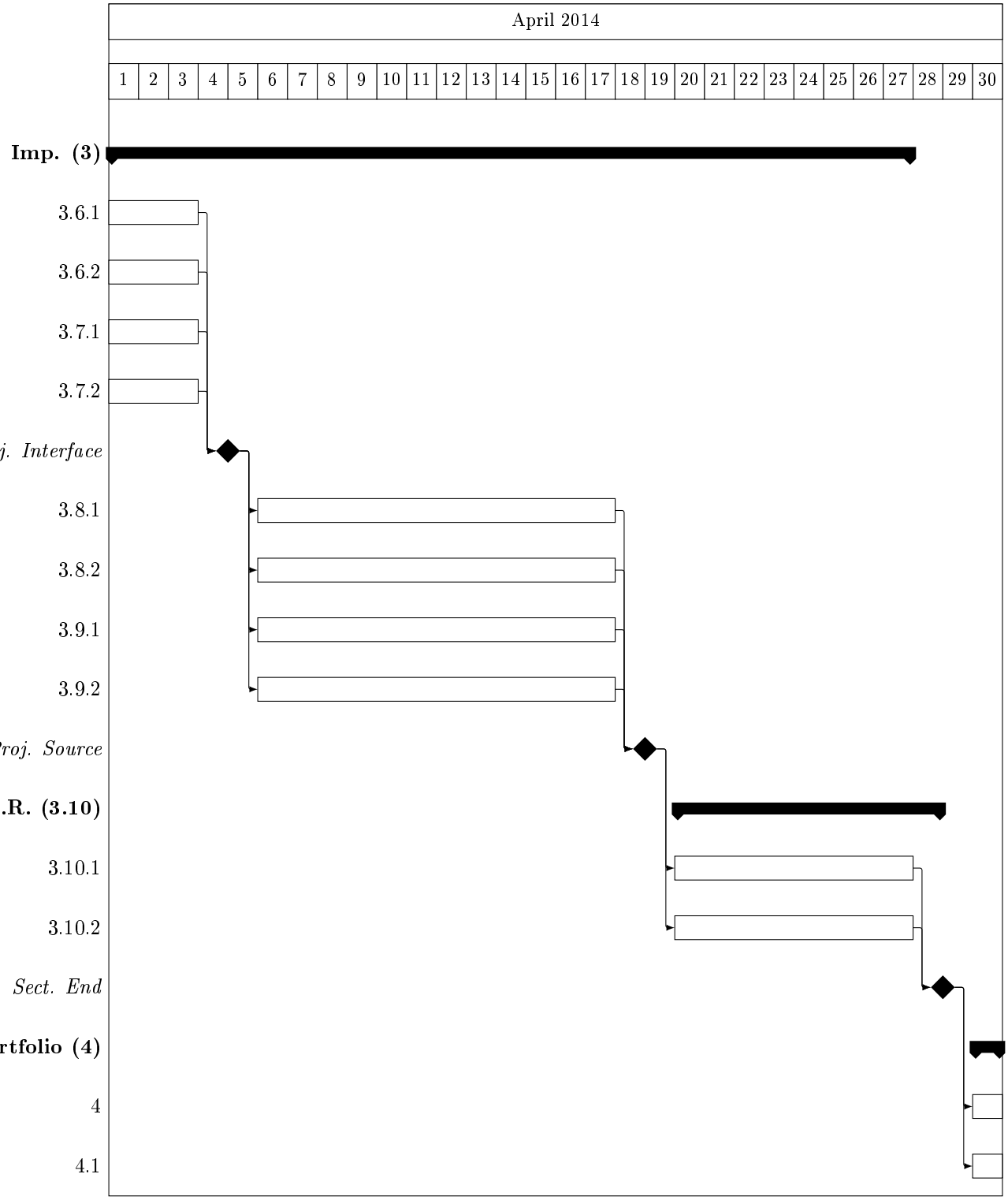
It is a graphical representation of the Task List documented prior to the charts. They have been split up either monthly or bi-monthly basis to allow acceptable formatting, due to differing workloads between months.

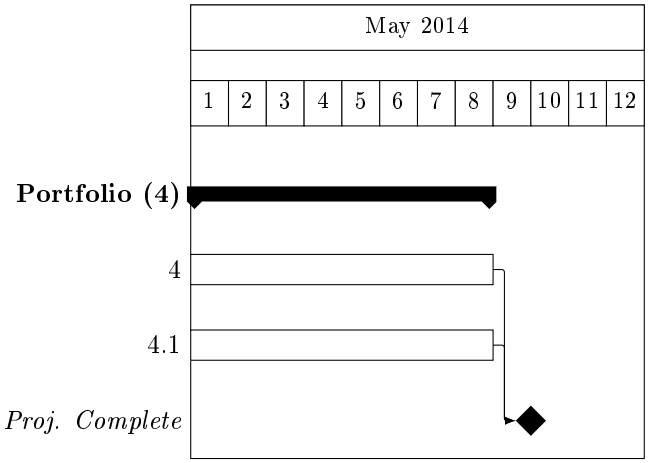
February is the most work intensive as it is providing the foundations of the project and is therefore detailed to reduce the risk of the project failing. April is the most lax of the working months (apart from May being project clean up) as the group members will be concentrating on the larger tasks as opposed to diverting attention between four or five tasks each, like in the Planning stage of the project.


February 2014
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Planning (1)
1.1
1.2
1.3
1.4
1.5
1.6
1.7
Scope
1.8.1
1.8.2
1.8.3
1.8.4
Case Studies
Technical Data
1.10
1.11
1.12
1.13
1.15
1.16
Planning Finished


February 2014
15
16
17
18
19
20
21
22
23
24
25
26
27
28
Design (2)
Research (2.1)
2.1.1
2.1.2
2.1.3
Research
sign Documentation(2.1)
2.2.1
2.2.2
2.2.3
2.2.4
2.2.5
2.2.6
2.2.7
2.2.8
2.2.9
2.2.10
2.2.11
2.2.12
2.2.13
Design Docs


March 2014
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Design (2)
S.R (2.3)
2.3.1
2.3.2
2.3.3
Sect. End
Imp. (3)
3.1.1
3.1.2
3.2.1
3.2.2
3.3.1
3.3.2
Proj. Backend
3.4.1
3.4.2
3.5.1
3.5.2
Protocols


April 2014
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Imp. (3)
3.6.1
3.6.2
3.7.1
3.7.2
j. Interface
3.8.1
3.8.2
3.9.1
3.9.2
roj. Source
.R. (3.10)
3.10.1
3.10.2
Sect. End
rtfolio (4)
4
4.1


May 2014
1
2
3
4
5
6
7
8
9
10
11
12
Portfolio (4)
4
4.1
Proj. Complete

Chapter 19

Risk Assessment

19.1 Parallel Tasks

A big concern for any project is the amount of tasks that will be performed simultaneously [**OracleWP**]. For every task that is carried out together, but potentially separate from each other, risk is increased - with more tasks making a more dramatic increase of potential failure for the project. For the planning section of this project we have performed a large amount of tasks simultaneously which may be detrimental to our quality of work later in the project.

In order to reduce or even eliminate the risk of too many parallel activities, the project should be planned using a Gantt chart to reduce the amount of tasks being performed simultaneously, and have more milestones within the project. This will help effectively split up the project into more manageable sections, which will not only make the project seem simpler to complete, but will improve the monitoring capabilities of the project as well.

19.2 Group Work

Working within a group can make deliverable dates difficult to achieve. This can be due to a lack of communication, unavailability of party members or an incapability to meet deadlines for some of the members. A meeting of minds also includes an assemblage of work ethics. Because of this, work may grind to a halt as members argue over personal yet trivial matters such as formatting documents or a varying opinion on what is classed as 'enough work' for a task.

Combating the disadvantages of working as a group can be difficult. As some problems are part of a group unable to function either properly of efficiently together, this can be the breaking factor of the project. This is a risk that cannot be eliminated but can be reduced. A way of minimising the amount of damage that the risk will do would be to have a centralised form of contacting members of

the group - examples being a website or using a revision control system such as 'Apache Subversion 'or 'Git', will give a common area for the group to look for potential absences or reasons for reduce work output from members.

The best way to reduce the risk of differing qualities of work between the group would be to define a standard of work between the group - such as the layout of source files in programming languages, or a house style for formal documents as part of the group's external identity. Having this be available to the group in some form, such as in a text file within a shared area will allow the group to refresh their memories of parts of the set standard that they wouldn't follow otherwise.

19.3 Deadlines

Deadlines are the final day or dates that an object needs to be completed by. Sometimes within a project the deadline may be overstepped due to any of the risks mentioned within this document, which can lead to something small such as being berated by the project leader or something serious such as a breach in contract with the client. For these reasons, deadlines need to be adhered to so that the project can continue on schedule.

Reducing the risk of deadlines are important, especially for those that are not capable of monitoring their time effectively. By providing deadlines as a range of dates as opposed to a singular date, there is increased flexibility within the project and it gives some people more time to finish their work if it is required. By using a range of dates the group can finish on the beginning of the deadline range - a sort of pseudo-deadline - meet up and discuss whether alterations need to be made on the work, and then use the remainder of the time until near the end of the deadline range to perform them.

19.4 Scope

The scope is what the project will be encompassing and therefore is one of the most important sections as it defines what you'll be doing for the entirety of the project. That's not the only risk associated with the scope [**RiskList**]. There is also scope creep, which is when the scope grows to cover more work than the project originally intended, often without an increase in resources matching the higher load on the project deliverables. Performing estimates on the scope, as well as anything else in the project, can be inaccurate as you are essentially guessing the near future, which is difficult at the best of times.

To minimise the risk placed upon the scope, it is best to define what exactly is required of the project before any work takes place on the deliverables. For example it is best to define an encryption method for a project at the beginning and sticking to it, rather than changing the method which may require a different implementation, creating more work. If at a later phase

ambiguities appear in the scope, a meeting to define or even redefine these points should occur before any more work is carried out on the offending article, reducing the amount of change to the project that shall occur.

19.5 Change Management

Change Management is the application of a structured process and set of tools for leading the people side of change to achieve a desired outcome [**changeManDef**]. Problems that are associated with Change Management include conflicts which occur between stakeholders, as they may be disagreeing in how the project should move forward. Assuming that an irreparable state has befell the project due to a drastic amount of changes that have been placed upon the project, or even ambiguous or inaccurate changes being added onto the project [**RiskList**]. All of these can amount into an increase in workload, or a decrease if the targets haven't been properly defined.

To reduce the amount of risk involved with Change Management, communication and clear definition on what the project needs to perform is required. Stakeholders should be as detailed as possible at every stage so that no ambiguity is caused, or cleared up if any does occur.

19.6 Stakeholders

Stakeholders are people that have an interest in the project, whether they are the members of the group, the group's monitor/superior or the target audience of the project. Some of the problems that Stakeholders cause for the project members include:

- Losing interest: if they become uninterested with the project then they may back out, which can be dangerous for the project if they were providing any form of input, such as experience in the target field or economic support.
- Stakeholders becoming disillusioned: They are unaware of what the deliverables will be or have a twisted view on what and how the final product will perform its intended purpose.
- Quality Risk: Stakeholders may give ambiguous input both accidentally or on purpose, depending whether the Stakeholder wants the project to fail or not [**RiskList**].

The best way to reduce the amount of risk involved with Stakeholders would be to keep them informed of the project's current status through external communication such as e-mail, and through meetings so that the team can personally inform the Stakeholder with relevant information, which should ease their mind of any apprehensive thoughts about the project [**stakeInfo**].

19.7 Platforms

The main risk in Platforms would be the difference between the chosen development platform and the target market's system. The change between executable files for different operating systems are usually great enough so that a separate executable is required for each distinct operating system. What may also cause problems, especially with low-level programming, would be differing architectures, hardware sets and how the system reads commands [**wikiArchDiff**]. Another problem with platforms would be whether the required software for the project is installed, such as any required run-time environments or files which are needed to use Structured Query Language databases.

This risk can be eliminated if platform-independent code is used - such as the Java Programming Language [**javaFeatures**]. This would mean that no changes in implementation would be needed, and database functionality could occur within the platform-independent environment if need be. Otherwise to reduce the amount of risk involved with the varying systems that the target audience may own, compiling the source on different virtual systems to create executables for the many various platforms available would suffice. Of course, this can be mitigated by choosing to not support other systems in favour of only allowing the development platform and its Operating System to be supported.

19.8 Integration

The integration of the project can be high risk due to a couple of factors:

- The intended environment is incompatible or unavailable
- Incomplete testing means the final product may be buggy
- Final product doesn't work (e.g. bad link to database)
- Product lowers efficiency due to learning curve [**RiskList**]

In order to combat the risks involved in implementation, having a set testing day in an isolated environment can allow the completed builds of the project to be evaluated before being given to the target audience. This will allow the checking of compatibility with the system as well as in-house bug testing. A manual or help section could be implemented into the system so that the learning curve is not as steep compared to not having such resources.

19.9 Requirements

Requirements are not just a list of functional needs and wants but also the constraints on the project as well. However, there are similar risks involved in the requirements, such as generalisation,

ambiguity or even being incomplete. Another risk to do with requirements is whether they align with the design factor or not.

An example would be having both 'fast processing' and 'system independence' as requirements; C++ is faster but Java is independent of platform and although speed may not be an issue with smaller data, larger chunks of data will undoubtedly have an effect on interpreted code [**javaCbenchmark**].

To minimise the risk with requirements, communication between group members and stakeholders is needed; making sure that the requirements and the scope are in line with each other, and that any suggested changes are properly handled with little to no ambiguity. Choosing a design structure and sticking to it is also beneficial to the project. Reducing the workload of the implementation can help towards minimising the risks of requirements and the program, such as removing old data that is no longer needed upon the program's start-up.

19.10 Authority

Without distinct authority within the project, risks can become apparent. If the members of the project do not have the correct privileges on the target system to perform what is required, work output slows or even stops until the matter is resolved. Another risk would be misguided authority; where the team is unclear who has been given the authority to perform a task and therefore there are multiple members allocating the same task to themselves, which will slow down the efficiency of the team due to duplicated work.

Lowering the negative impact of Authority is done through the use of clear definitions. Allocating work to project members and centralising a form of 'to-do' list so that project members can look up what has been assigned to them. Another way of reducing the amount of inefficiency caused by problems with authority would be to make sure the permissions are correctly set up on both the testing and target systems.

19.11 External

There are a couple of external factors which may impact the project in a negative manner. The first being any legal restrictions. This is important as there is a chance that the final product may be used in a location that differs to the geographical area that it was developed in. For example there is a law within the UK which requires that you must provide encryption keys under certain circumstances to the UK authorities [**ripaIII**][**ukCryptLaw**]. In the USA however, it is something of a grey-area, as giving up encryption keys could violate the fifth amendment, as doing so could give incriminating evidence against yourself:

> 'unlike surrendering a key, disclosing a password reveals the contents of oneâĂŹs mind and is therefore testimonial.' [**usCryptLaw**]

Not only is the law a big risk in projects, but also nature. If you are situated where natural disasters can happen or otherwise things such as heavy weather occur, this can reduce the work flow by denying the team members access to their workspace. Another factor that is external is the changing of technology. Updates to programming languages can lead to deprecated functions or newer operating systems may not be capable of running the same software as their previous iterations, meaning an increased amount of work to keep the software compatible with the target system.

Reducing the amount of risk caused by external factors is difficult as the project team have little to no influence upon them. For example the team cannot bypass any laws that govern the area that the program will be used in, so they must be adhered to as part of the constraints of the project. Natural disaster cannot be stopped, but if you are able to, bringing some of the work back so you could work on it during bad weather may reduce the impact that said weather will have on the project. To reduce the damage caused by software deprecation it is ideal if the functionality coded in the project is not old, or otherwise buggy, so that maintaining or updating the software will require less work.

19.12 Project Management

Project Management, or rather a lack of, can also be a risk to the endeavours of the team. If the group has been asked to reduce or combine the amount of stages in the System Development Life Cycle (SDLC), this can increase the risk of the project failing because it leaves more room for error; combining the stages will often cause a decrease in quality, as less resources are being dedicated to a particular section of the project. A lack of Project Management will also be seen as a high risk because of how difficult it is to monitor a project and its success without these tools.

To reduce the risk that Project Management will apply upon the project, a formal methodology, such as the 'waterfall' method could be implemented. This would however reduce inefficiency as the output needs to be moderated and cleared before the start of the next stage in the SDLC can occur. On the other hand an informal methodology would increase the risks, but may potentially allow the project to be completed within a smaller time frame and to the same standard.

19.13 User Acceptance

Just because a project has been made for a target audience doesn't mean that *that* audience will like it. During testing the target market may reject the initial builds of the project due to the way

it does or does not work, or the look of the project could mean that it is unwieldy to use, whether it is due to low quality or the interface being anti-intuitive.

The main method of reducing the risk pre-emptively is to perform research on any currently available software that achieve similar goals to the project's. By doing this you can find out what users are acquainted with and create a similar yet unique design, or use the competitors as a way of highlighting what is wrong with the current market and create something entirely different. Another method which does require more work is to take in user feedback during testing and implement their suggestions for the look of the project, or the inner mechanics if they have the knowledge to suggest improvements.

19.14 Conclusion

In order to reduce the risk of the project as a generalisation, it is suggested that you:

- Have a centralised communication system used by all members - this reduces all communicative related risks.
- Define team objectives and allocation clearly - this reduces the authority-based risks as well as any that are communicative.
- Define a target system for development - other types of platform can be supported at a later date should the need arise.
- Create and uphold a work ethic to be followed by everyone - this helps to maintain a standard of quality throughout the project.
- Testing should be first on each individual module/deliverable, then as a whole. This improves bug catching and helps monitor the quality of the project.
- Choose a methodology and follow it - this creates a standard of work ethics which will give a layout as well as structure to the project.

By following these pointers a moderate amount of risk can be mitigated with little need for concern. Do note that the legality of the project in differing countries should be researched and followed, should the project be in use within that country.

Part II

Design

Chapter 20

Proposal Summary

The project, a security based social media network, will have multiple components to be investigated and used in this design section. The key critical components to be looked at consist of:

- Database
- Client
- Client GUI
- Server
- Server GUI
- Mobile GUI (future work)

Of each of these components we should look at how they will impact their respective uses in order to best make use of their full functionality. We will look at multiple possible and practical solutions for the above criteria, making sure the best solution is chosen. We will also look at possible work in the future, or any areas to continue with into the coming stages.

The requirements section has helped so far through analysis of existing social media networks and how they have implemented their networks, along with how their interfaces react to the user.

Chapter 21

Architecture

21.1 Network Architecture

Turtlenet is a centralized service, whereby a large number of clients connect to a single server which provides storage, and facilitates communication between clients.

Due to the inherently limited network size (5-50K users per server depending on percentage of active participants vs consumers and local internet speeds) we recommend that servers serve a particular interest group or geographic locality.

Clients send messages to, and only to, these central servers. Due to the fact that all messages (except CLAIM messages, see client-server/client-client protocols for details) are encrypted the server does not maintain a database, it cannot; rather clients each maintain their own local database, populated with such information to which they have been granted access.

When a client wishes to send a message to a person, they encrypt the message with the public key of the recipient[1] and upload it to the server. It is important to note that all network connections are performed via Tor.

When a client wishes to view messages sent to them, they download all messages posted to the server since they last downloaded all messages from it, and attempt to decrypt them all with their private key; those messages the client successfully decrypts (message decryption/integrity is verified via SHA256 hash) were intended for it and parsed. During the parsing of a message the sender is determined by seeing which known public key can verify the RSA signature.

Due to the nature of data storage in client-local databases, all events and data within the system must be represented within these plaintext messages. This is achieved by having multiple types of messages (see client-client protocol).

[1] using RSA/AES, see protocol for details

21.2 System Architecture

The system has a number of modules which interact with one another via strictly defined interfaces. Each module has one function, and interacts as little as possible with the rest of the system. The modules and their interactions are shown below. NB: a->b denotes that data passes from module a to module b, and a<->b similarly denotes that data passes both from a to b and from b to a.

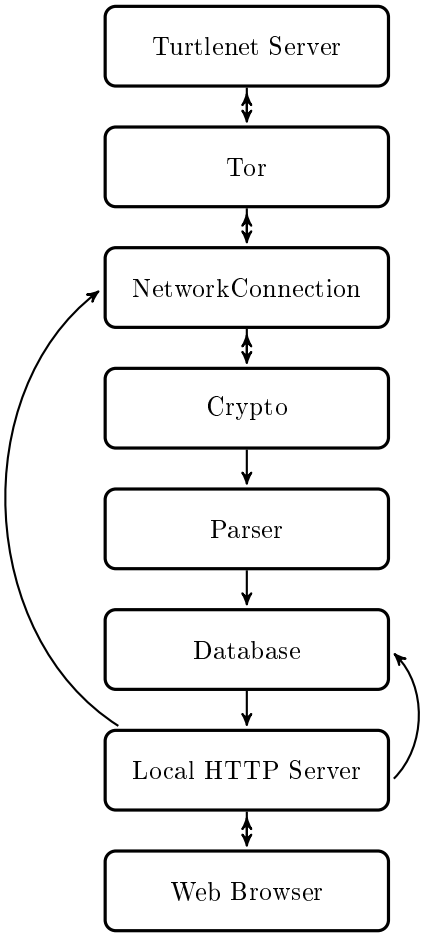


Figure 21.1: Module Interaction

21.3 Data Flow Diagram

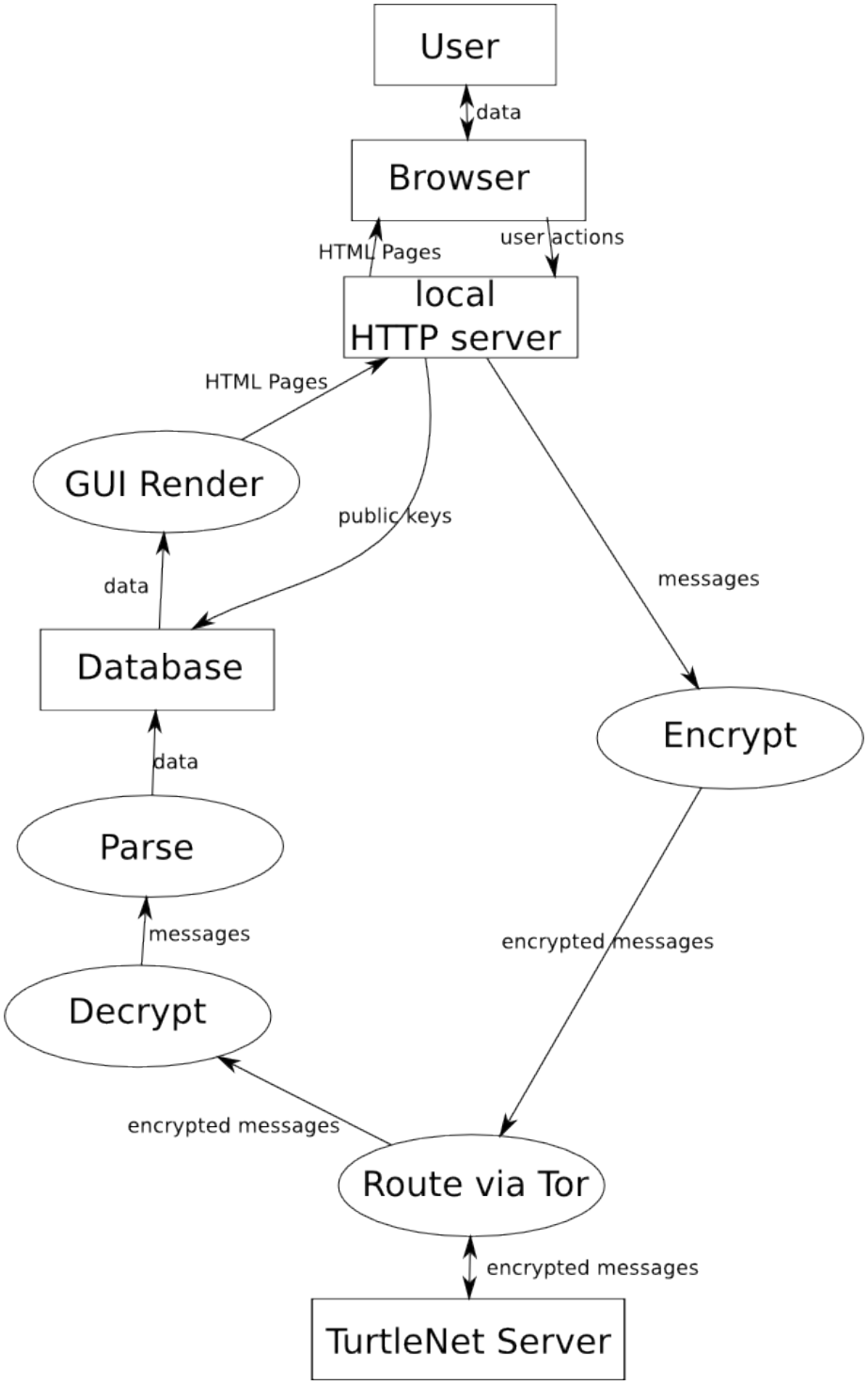
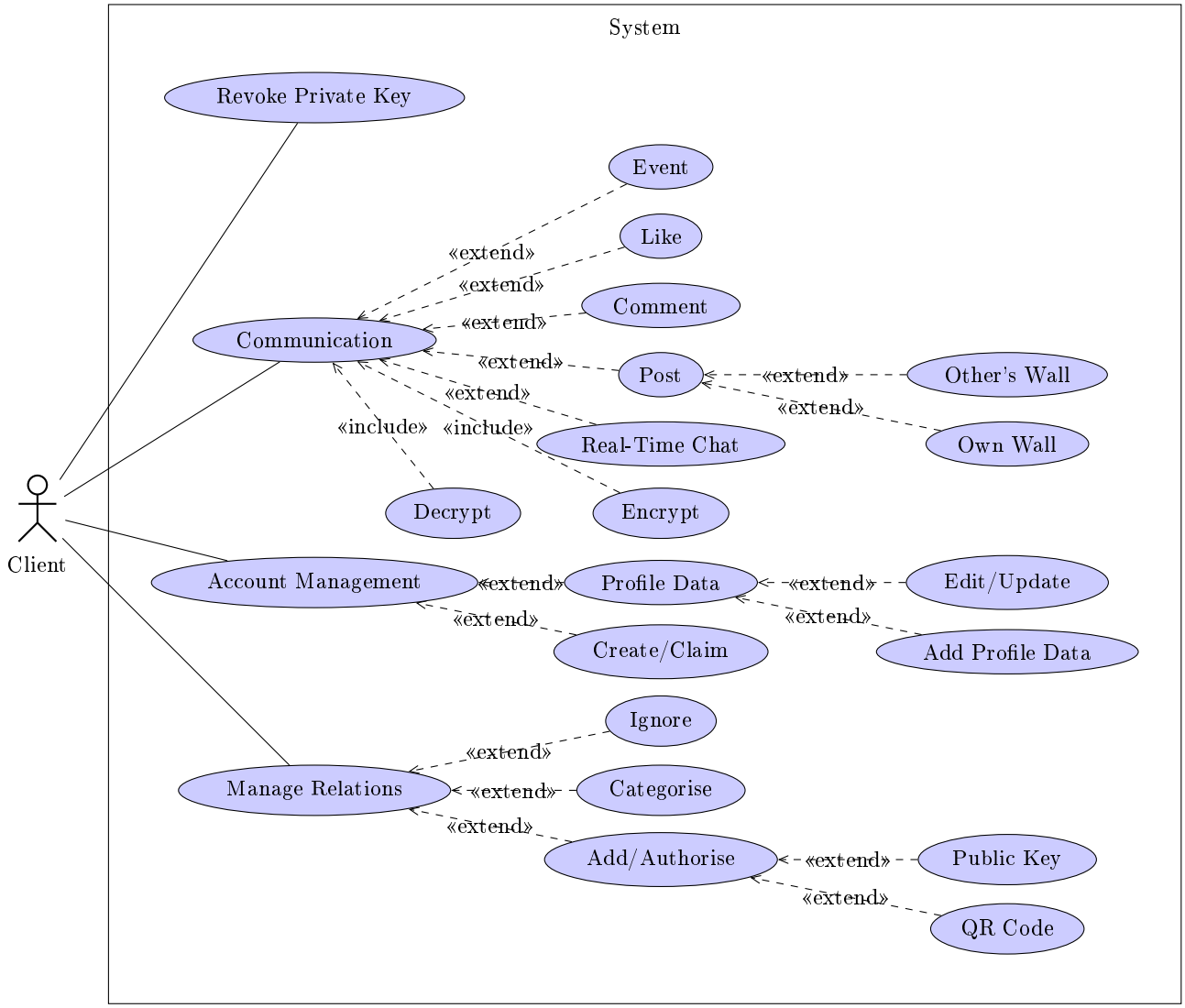


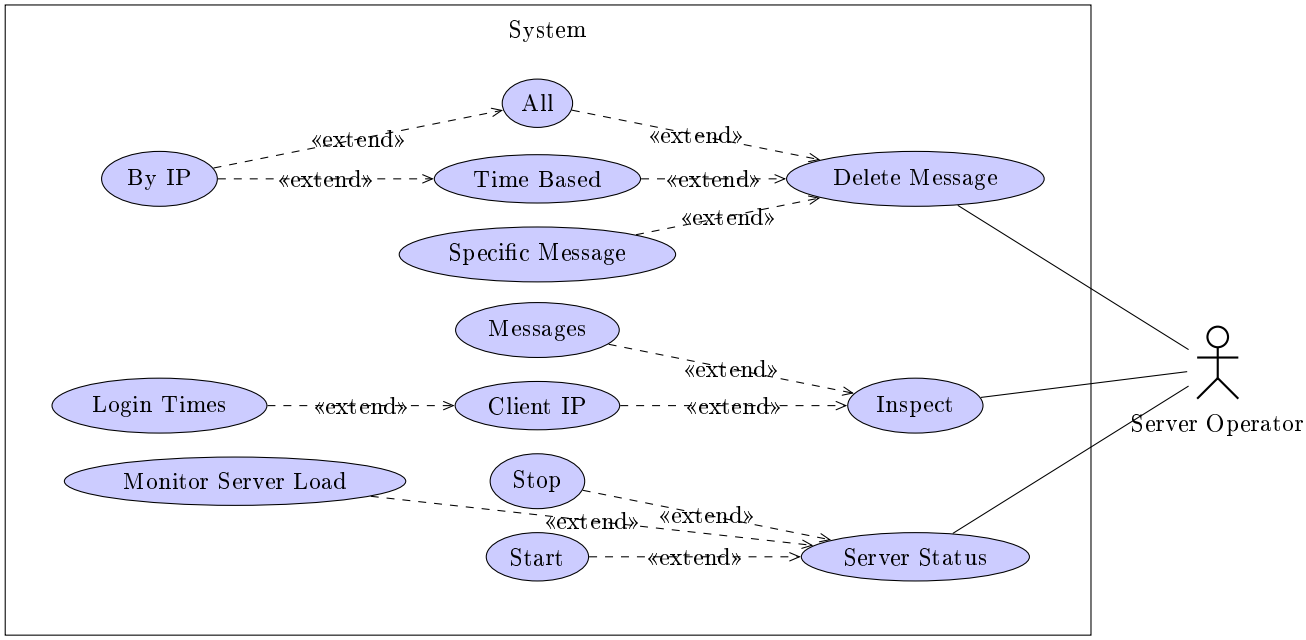
Figure 21.2: Data Flow Diagram

Chapter 22

Use Case Diagrams

Here we have a use case diagram displaying an actors interaction with our system. It shows the functionality available to both the client, and the operator. We also have a sequence diagram to augment the use case diagrams.


System
Revoke Private Key
Event
Like
Comment
Communication
Post
Other's Wall
Own Wall
Real-Time Chat
Decrypt
Encrypt
Client
Account Management
Profile Data
Edit/Update
Create/Claim
Add Profile Data
Ignore
Manage Relations
Categorise
Add/Authorise
Public Key
QR Code
«extend»
«include»


System
All
By IP
Time Based
Delete Message
Specific Message
Messages
Login Times
Client IP
Inspect
Monitor Server Load
Stop
Start
Server Status
«extend»
Server Operator

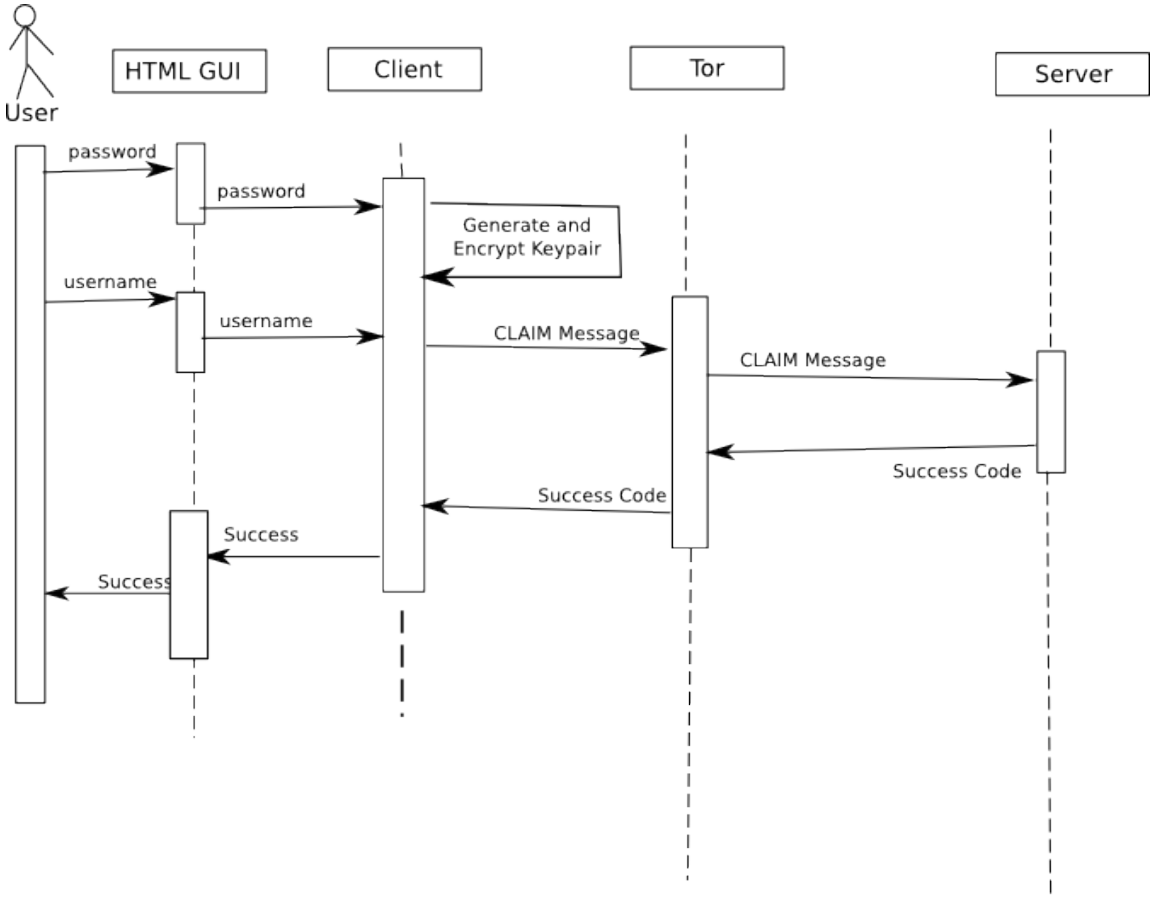


Figure 22.1: Sequence Diagram - Registering

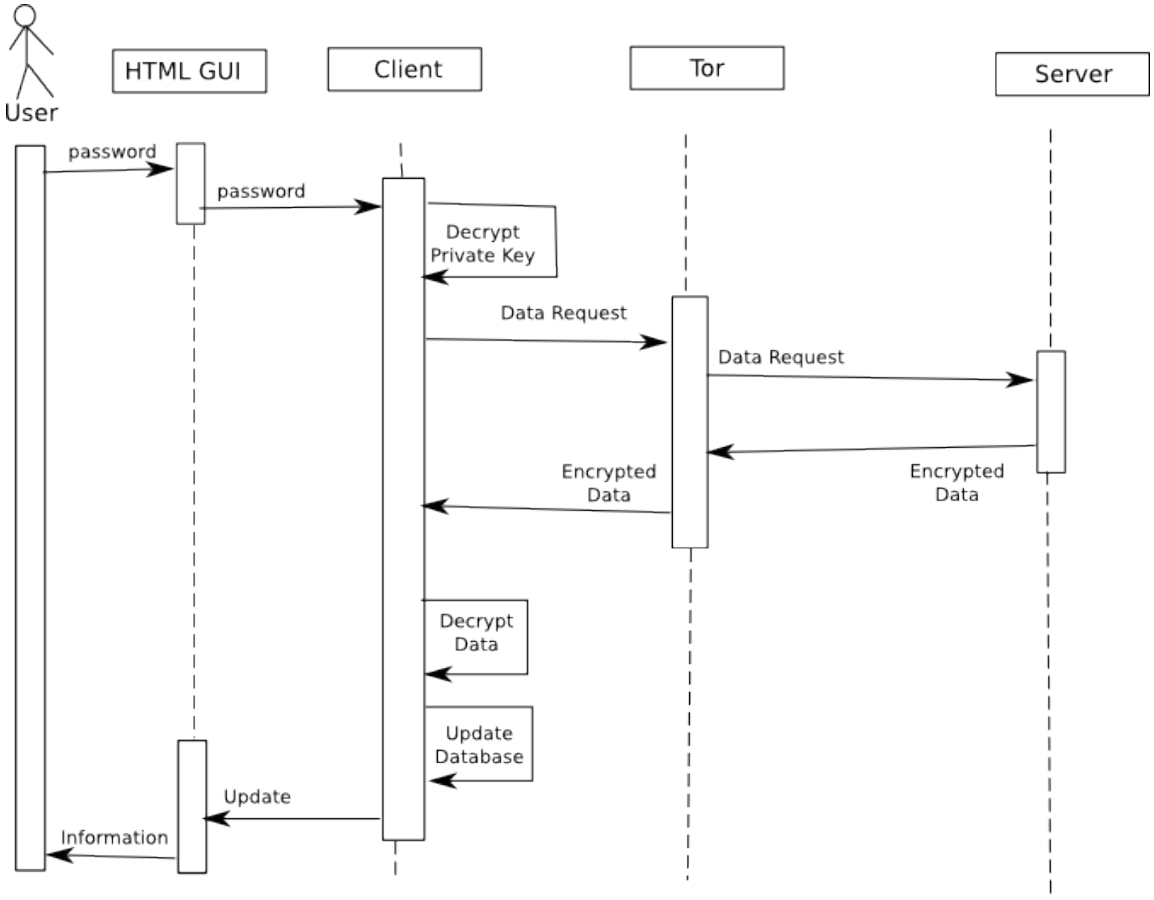


Figure 22.2: Sequence Diagram - Recieving Data

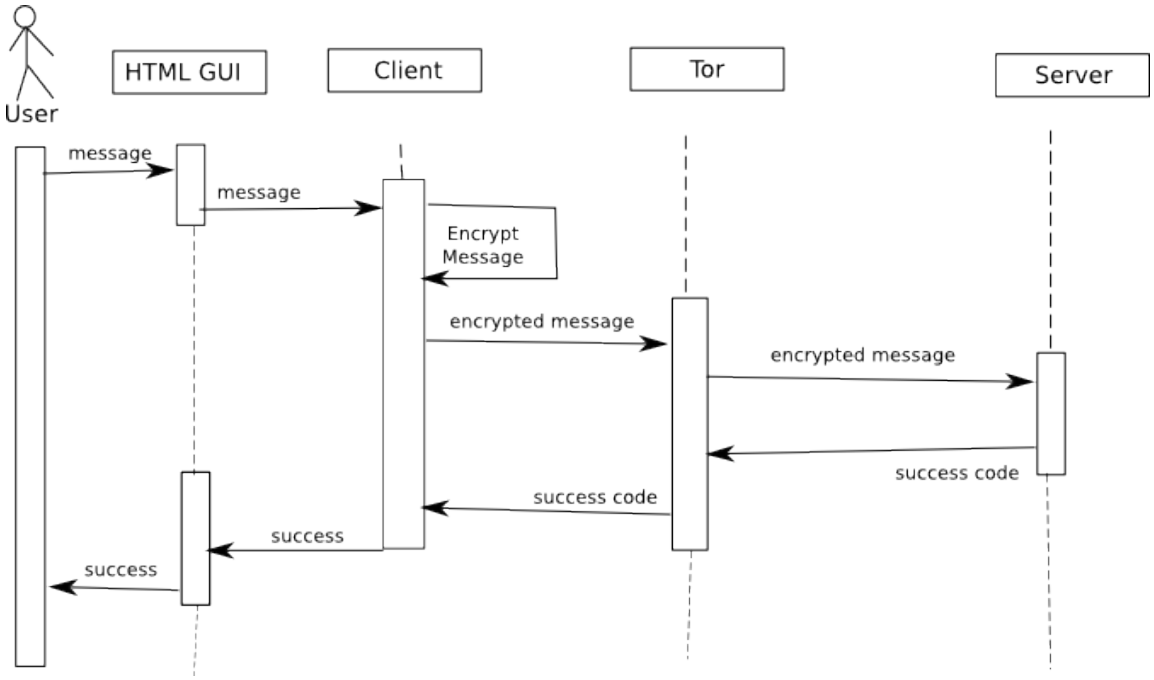


Figure 22.3: Sequence Diagram - Sending Data

Chapter 23

Protocol

23.1 High Level Summary of Protocol

Creating an account is done by generating an RSA keypair, and choosing a name. An unencrypted (but signed) message is then posted to the server associating that keypair with that name. In this way, by knowing the public key of someone, you may discover their name in the service, but not vice versa.

Connecting for the first time Every unencrypted message stored on the server is downloaded(signed nicknames and nothing more). At this time the local database contains only signed messages claiming usernames. The public keys are not provided, these are of use only when you learn the public key behind a name. The rationale for not providing public keys is provided in the section regarding adding a friend. Messages posted after your name was claimed will require downloading too, as once you claim a name people may send you messages. It's worth noting that messages from before you connected for the first time are now downloaded because they can not have been sent to you (with a compliment client) if someone retroactively grants you permission to view something they publish it as a new message with an old timestamp; the sole exception to this is when you connect using a new device, in which case all messages since you first claimed a name will be downloaded.

Connecting subsequently The client requests every message stored on the server since the last time they connected up to the present. Decryptable messages are used to update the local DB, others are discarded.

Continued connection During a session the client requests updates from the server every

0.5-5 seconds (configurable by the user).

Adding a friend is performed by having a friend email (or otherwise transfer) you their public key. This is input to the client, and it finds their username (via public posting that occurred when registering). You may now interact with that person. They may not interact with you until they receive your public key. Public key transferral will be performed via exchanging plaintext base64 encoded strings, or QR codes. The user will be prompted, after retrieving the username of the user, to categorise them.

Talking with a friend or posting on your wall is achieved by writing a message, signing it with your private key, and encrypting one copy of it with each of the recipients public keys before posting it to the server. The client prevents one from posting a message to someone's public key if they have not claimed a nickname.

Posting to a friends wall, commenting and liking may be requested by sending a EPOST/CMNT/LIKE message to the friend (upon whose wall/post you are posting, commenting or liking), when that friend logs in they will receive your request, and may confirm or deny it. If they confirm then they take your (signed) message and transmit it to each of their friends as previously described. Given that authentication is entirely based on crypto signatures it doesn't matter that your friend relays the message.This is required because it is impossible for one to know who is able to see the persons wall, post, or comment upon which you seek to post, like, or comment.

23.2 Client-Server Protocol

The client-server architecture is necessarily simple.

The client connects to the server, sends a single command, receives the servers response and then disconnects. The following shows commands sent by the client, and the servers action in response.

command	purpose	servers action
t	get the server time	sends back the current time (unix time in milliseconds)
s *utf-8_text*	send messages	the text sent is stored on the server
get *ms_unix_time*	get new messages	every message stored since the given time is sent
c *utf-8_text*	claim a username	the text sent is stored on the server, with a special filename

Table 23.1: Client-Server Protocol

Every command is terminated with a linefeed. Every response from the server will be terminated with a linefeed. The last line sent by the server will always be "s" for success, or "e" for failure (this is omitted from the above table).

CLAIM messages (sent with c) will be parsed by the Message class and the username extracted for use in a filename. The filename of claim messages is as follows <*unix_time_in_ms*>_<*username*>; the filename of all other messages is as follows <*unix_time_in_ms*>_<*SHA256_hash*>.

23.3 Client-Client Protocol

23.4 Summary

All client-client communication is mediated by the server. When one client wishes to send a message to another it encrypts the message with the public key associated with the recipient and uploads it to the server. When one client wishes to receive a message it downloads all new messages from the server and parses those it can decrypt. This is performed in order to hide who receives a message. All messages except CLAIM messages are encrypted. Multiple recipients imply multiple messages being uploaded, this is taken for granted in the text which follows.

23.5 Message Formatting

23.5.1 Unecrypted Messages

Messages have a command (or type), which specifies the nature of the message; messages have content, which specifies the details of the message; messages have an RSA signature, which authenticates the message; messages have a timestamp, which dates the message down to the millisecond, the time format is unix time in milliseconds.

Messages are represented external to the system as utf-8 strings, and internally via the Message class. The string representation is as follows:

<*command*>\<*signature*>\<*content*>\<*timestamp*>

Backslashes are literal, angle brackets denote placeholder values where data specific to a message is placed.

An example follows:

POST\<*signature*>\Hello, World!\1393407435547

backslashes in message content are escaped with another backslash, signatures are base64 encoded SHA256/RSA signatures of the content of the message concatenated with a decimal string representation of the timestamp. All text is encoded in UTF-8.

23.5.2 Encrypted Messages

Encrypted messages contain the AES IV's; the RSA encrypted AES key; and the AES encrypted message.

Messages are encrypted by encoding the entire message to be sent with UTF-8; encrypting the message with a randomly generated AES key; encrypting the AES key with RSA; encoding the RSA encrypted AES key in base64; encoding the (random) AES initialization vectors in base64 and concatenating these three parts with a backslash between each. The format follows:

$$<AES\ IV>\backslash<RSA\ encrypted\ random\ AES\ key>\backslash<AES\ encrypted\ message>$$

Backslashes are literal, angle brackets denote placeholder values where data specific to a message is placed.

23.6 Claiming a Username

Each user (keypair) should claim one username. Uniqueness is enforced by the server, and so not relied upon at all. Usernames are useful because public keys are not human readable. In order to claim a username, one must sent an unencrypted CLAIM message to the server. The format follows:

$$\text{CLAIM}\backslash<signature>\backslash<username>\backslash<timestamp>$$

23.7 Revoking a Key

If a users private key should be leaked, then they must be able to revoke that key. This is done by sending a REVOKE message to the server. All content signed by the private key after the stated time will be flagged as untrusted. The format follows:

$$\text{REVOKE}\backslash<signature>\backslash<time>\backslash<timestamp>$$

23.8 Profile Data

Users may wish to share personal details with certain people, they may share this information via profile data. Profile data is shared using PDATA messages. A PDATA message contains a list of fields, followed by a colon, followed by the value, followed by a semicolon. The format follows:

$$\text{PDATA}\backslash<signature>\backslash<values>\backslash<timestamp>$$

The format for values follows:

$$<field>: <value>; \ldots$$

An example follows:

$$\text{PDATA}\backslash<signature>\backslash\text{name:Luke Thomas;dob:1994;}\backslash<timestamp>$$

23.9 Inter-User Realtime Chat

Users can chat in in real time, this by achieved by sending a CHAT message to all people you wish the include in the conversation. This message includes a full list of colon delimited public keys involved in the chat. The format follows:

$$\text{CHAT}\backslash<signature>\backslash<keys>\backslash<timestamp>$$

The format for keys follows:

$$<key>: <another\_key>\ldots$$

An example follows:

$$\text{CHAT}\backslash<signature>\backslash<key1>:<key2>\backslash<timestamp>$$

Following the establishment of a conversation, messages may be added to it with PCHAT messages, the format follows:

$$\text{PCHAT}\backslash<signature>\backslash<conversation>:<message>\backslash<timestamp>$$

Whereby $<conversation>$denotes the signature present on the establishing message. An example follows:

$$\text{PCHAT}\backslash<signature>\backslash\text{9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08:First!}\backslash<timestamp>$$

23.10 Posting to own wall

When a user posts to their own wall they upload a POST message to the server of the following format.

$$\text{POST}\backslash<signature>\backslash<message>\backslash<timestamp>$$

The format of message is merely UTF-8 text, with backslashes escaped with backslashes.
An example follows which contains the text "Hello, World!", a newline, "foo \bar\baz":

$$\text{POST}\backslash<signature>\backslash\text{Hello, World!}$$
$$\text{foo}\backslash\backslash\text{bar}\backslash\backslash\text{baz}\backslash<timestamp>$$

23.11 Posting on another users wall

A user may request to post on a friends wall by sending them an FPOST message, the poster may not decide who is able to view the message. The format is identical to that of a POST message, except for the command and singular recipient. An example follows:

FPOST\<*signature*>\Hello, World!\<*timestamp*>

Upon receipt of an FPOST message the friend is prompted by the client to choose whether or not to display it, and if so who may view it. Once this is done the friend reposts the message with the command changed to POST instead of FPOST as they would post anything to their own wall. This works because authentication is entirely based on RSA signatures so in copying the original signature the friend may post as the original author provided they don't alter the message (and thus its hash and required signature).

23.12 Commenting

Commenting works similarly to posting on another's wall, so an explanation of details of how it occurs is not provided (see prior section). The only difference is the format of a CMNT message from an FPOST message. The format of a CMNT message is as follows:

CMNT\<*signature*>\<*hash*>: <*comment*>\<*timestamp*>

Where <*hash*>denotes the hash of the post or comment being commented upon. An example comment follows:

CMNT\<*signature*>\v/sXfb3DG2qT2k2hXIH4csJy1yEG+TANRbbxQw1VkSE=: Yeah, well, that's just like, your opinion, man.\<*timestamp*>

23.13 Liking

Like messages are identical to comments except for the command and the the fact that no ":<*comment*>" follows the hash. An example like follows:

LIKE\<*signature*>\v/sXfb3DG2qT2k2hXIH4csJy1yEG+TANRbbxQw1VkSE=\<*timestamp*>

23.14 Events

A user may have the client remind him of an event by alerting him when it occurs. A user may inform others of events, and they may choose to be reminded about them. When a user creates

an event just for themselves they just create a normal event and only inform themselves of it. An event is created by posting an EVNT message to the server. The format follows:

EVNT\<*signature*>\<*event_start_time*>: <*event_end_time*>: <*event_name*>\<*timestamp*>

An example follows of a reminder for bobs birthday which occurs on the 14th of January, the event was created on the second of January:

EVNT\<*signature*>\1389657600000: 1389744000000: bobs birthday\1388676821000

Chapter 24

Class Interfaces

24.1 Class Interfaces

The following is a description of the public functions of all public classes. Many classes have inner private classes they use for convenience, however to simplify interaction between parts of our system ('modules') we have very few convenience classes.

return	function	description
void	main()	(static) starts the server

Table 24.1: Server

return	function	description
void	main()	(static) constructs and starts all necessary classes and threads, runs the main loop

Table 24.2: Client

return	function	description
N/A	NetworkConnection()	Constructs a NetworkConnection and connects to the given URL (through tor)
void	run()	periodically download new messages until asked to close, downloaded messages are stored in a FIFO buffer
void	close()	kills the thread started by run()
boolean	hasMessage()	return true if there is a message in the buffer, false otherwise
String	getMessage()	return the oldest message in the buffer
boolean	claimName()	claim a given username, returns true on success, false otherwise
void	revokeKeypair()	revokes your keypair
void	pdata()	adds or updates profile information
void	chat()	begins or continues a conversation
void	post()	post a message to your wall
void	fpost()	post a message to a friends wall
void	comment()	comment on a comment or post
void	like()	like a comment or post
void	event()	create an event

Table 24.3: NetworkConnection

return	function	description
boolean	keysExist()	(static) return true if the user has a keypair, false otherwise
void	keyGen()	(static) generate a keypair for the user
PublicKey	getPublicKey()	(static) returns the users public key
PrivateKey	getPrivateKey()	(static) returns the users private key
String	sign()	(static) returns an RSA signature of the passed string
boolean	verifySig()	(static) returns true if author signed msg, false otherwise
String	encrypt()	(static) returns an encrypted message constructed from the passed parameters
Message	decrypt()	(static) decrypts the passed string, returns the appropriate message, on failure a NULL message is returned
String	base64Encode()	(static) base64 encodes the passed data, returns the string
byte[]	base64Decode()	(static) base64 decodes the passed data, returns the byte[]
String	encodeKey()	(static) encodes a public key as a string, returns that string (X509)
PublicKey	decodeKey()	(static) decodes a public key encoded as a string, returns that public key(X509)
String	hash ()	(static) returns the SHA256 hash the the passed string as a hex string
int	rand ()	(static) returns a pseudorandom value <= max and >= min

Table 24.4: Crypto

return	function	description
void	parse()	(static) parses a sting message, records parsed data in the database

Table 24.5: Parser

return	function	description
void	addClaim()	adds a username CLAIM message
pair<string,string>[]	getClaims()	gets all CLAIMs to usernames
string[]	getUsernames()	gets all usernames
void	addRevocation()	adds a keypair revocation
pair<PublicKey, long>[]	getRevocations()	gets all revocations
boolean	isRevoked()	returns the time a key was revoked, if the given key has not been revokes then 0 is returned.
void	addPData()	adds (or amends existing) profile data
string	getPData()	gets the specified piece of profile data for a specified user
void	createChat()	creates new chat
pair<string,string>[]	getChat()	returns messages from a given chat
void	addToChat()	adds a post to a given chat
void	addPost()	creates new post, on your or another's wall
pair<string,string>[]	getPosts()	gets all posts either within timeframe, or from certain people within a timeframe
void	addComment()	adds a comment onto post or comment
pair<string,string>[]	getComments()	gets all comments for a post or comment
void	addLike()	likes a post or comment
String[]	getLikes()	gets all likes from certain person within a timeframe
int	countLikes()	gets the number of likes for a comment or post
void	addEvent()	adds new event
pair<string,long>[]	getEvent()	gets all events within timeframe
void	acceptEvent()	accepts notification of an event
void	declineEvent()	declines notification of an event
void	addKey()	adds a public key to the DB
PublicKey[]	getKey()	gets the public key for a usernamne, or all which are stored)
string	getName()	gets a username for the given public key
void	addCategory()	adds a new category to the DB
void	addToCategory()	adds a user to a category

Table 24.6: Database

return	function	description
N/A	GUI()	Constructs a GUI
void	run()	continually updates the GUI from the DB
void	close()	kills the GUIServer thread
boolean	isRunning()	returns true if the GUIServer is running, false otherwise

Table 24.7: GUI

return	function	description
N/A	Message()	Constructs a message with given data
Message	parse()	(static) parses the string representation of a message into a message
String	toString()	creates a string representation of the message
String	getCmd()	returns the type of message
String	getContent()	returns the content of the message
String	getSig()	returns the RSA signature on the message
long	getTimestamp()	returns the timestamp on the message

Table 24.8: Message

return	function	description
N/A	Pair()	Constructs a pair with given data
A	first()	returns the first value passed to the constructor
B	second()	returns the second value passed to the constructor

Table 24.9: Pair<A, B>

24.2 Class Diagram

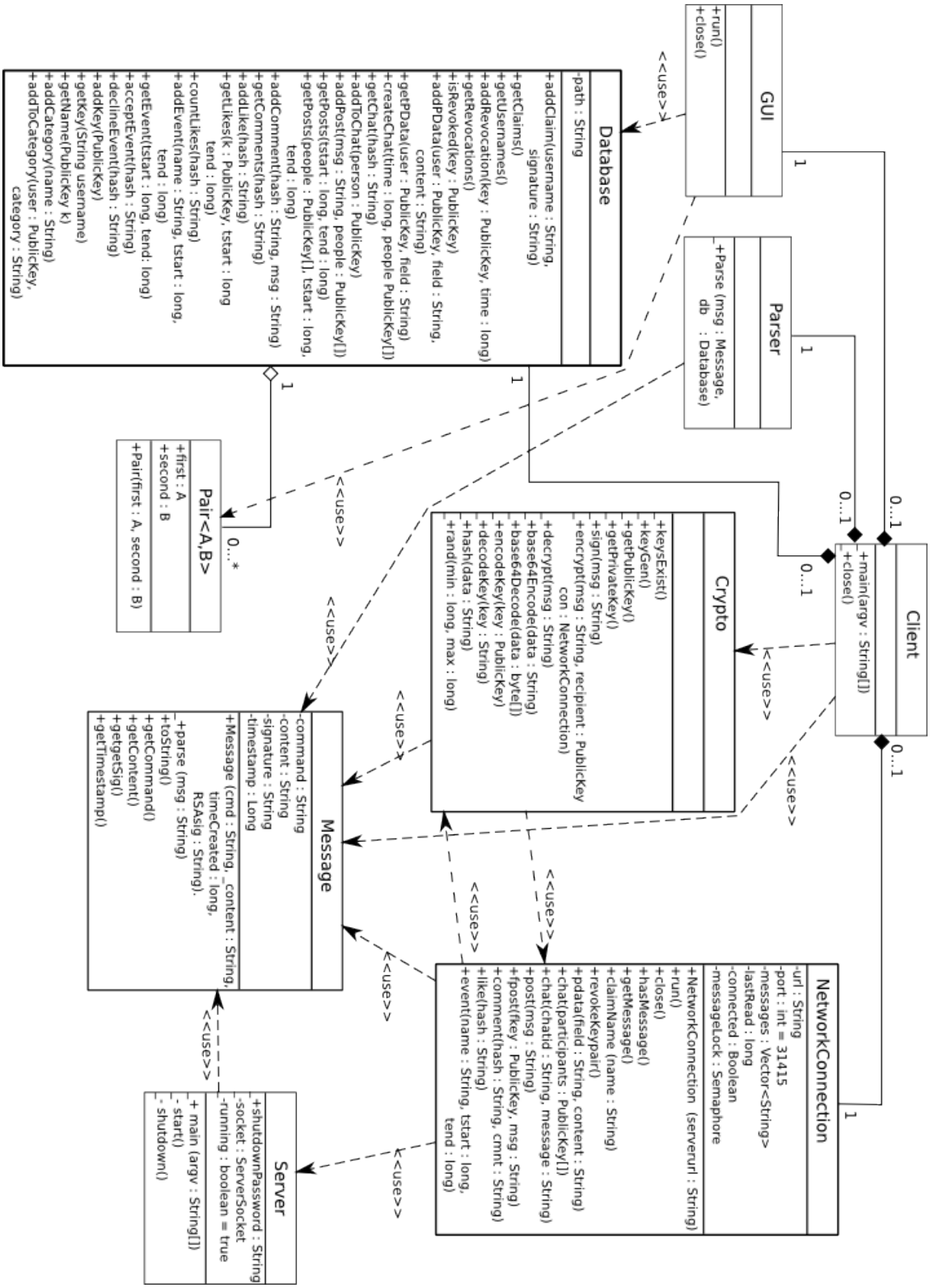


Figure 24.1: UML Class Diagram

Chapter 25

Pseudocode

25.1 Server

```
static void main () {
    startGUIthread()
    startServer()
}

static void start () {
    socket = new ServerSocket(port)
    while (running) {
        incoming = socket.accept()
        t = new Thread(new Session(incoming))
        t.start()
    }

    shutdown()
}
```

25.2 Client

```
static void main () {
    NetworkConnection connection    = new NetworkConnection("server.tld")
    Thread            networkThread = new Thread(connection)
    Database          db            = new Database("./db")
```

```
    GUI                 gui             = new GUI(db, connection)
    Thread              guiThread       = new Thread(gui)

    if (!Crypto.keysExist())
        Crypto.keyGen()

    networkThread.start()
    guiThread.start()

    while (gui.isRunning())
        while (connection.hasMessage())
            Parser.parse(Crypto.decrypt(connection.getMessage()), db)
}
```

25.3 Crypto

```
keyGen () {
    keypair = generateRSAkeypair()
    pw      = GUI.getUserInputString()
    filesystem.write("keypair", Crypto.aes(pw, keypair))
}

static String sign (String msg) {
    byte[] sig = SHA1RSAsign(msg.getBytes("UTF-8"), Crypto.getPrivateKey())
    return Crypto.Base64Encode(sig)
}

static String encrypt(String cmd, String text, PublicKey recipient,
                      NetworkConnection connection) {
    Message msg = new Message(cmd, text, connection.getTime()+Crypto.rand(0,50),
                              Crypto.sign(text))

    //encrypt with random AES key with random initalization vectors
    byte[]      iv = new byte[16]
    byte[] aeskey = new byte[16]

    fillWithRandomData(iv);
    fillWithRandomData(aeskey);
```

```
    byte[] aesCipherText = aes(aeskey, iv, msg.toString().getBytes("UTF-8"))

    //encrypt AES key with RSA
    byte[] encryptedAESKey = rsa(Crypto.getPrivateKey(), aeskey)

    //"iv\RSA encrypted AES key\ciper text"
    return Base64Encode(iv) + "\\" + Base64Encode(encryptedAESKey) +
           "\\" + Base64Encode(aesCipherText)
}

static Message decrypt(String msg) {
    //handle claim messages (which are the only plaintext in the system)
    if (msg.substring(0,2).equals("c "))
        return Message.parse(Base64Decode(msg.substring(2)))

    //handle encrypted messages
    String[] tokens = new String[3]
    tokens = tokenize("msg", "\")

    byte[] iv           = Base64Decode(tokens[0])
    byte[] cipheredKey  = Base64Decode(tokens[1])
    byte[] cipherText   = Base64Decode(tokens[2])

    //decrypt AES key
    byte[] aesKey = rsaDecrypt(cipheredKey, getPrivateKey())

    //decrypt AES Ciphertext
    aes.init(Cipher.DECRYPT_MODE, aesKeySpec, IVSpec)
    byte[] messagePlaintext = aesDecrypt(cipherText, aesKey, iv)

    return Message.parse(messagePlaintext)
}
```

25.4 Database

Most database functions are just going to construct parameterized SQL queries to be sent to the database from passed parameter values. The exceptions which include significant computing are

listed here:

```
void addKey (PublicKey k) {
    for each row r in table message_claim
        if (Crypto.verifySig(r.signature, k))
            addFriend(new Friend(k, r.username))
}

PublicKey[] getKey (String username) {
    PublicKey[] keys
    for each row r in table user
        if (r.username == username)
            keys.add(r.public_key)
    return keys
}

void addToCategory (Friend f, String category) {
    for each row r in table wall_post
        if (r.permission_to includes category)
            sendMessage(r, f)
}
```

25.5 Network Connection

The vasy majority of messages here merely construct the appropriate message from the parameters and pass it to serverCmd()

```
void main (String _url) {
    url = _url
    messages    = new Vector<String>()
    messageLock = new Semaphore(1)
    connected   = true

    File lastReadFile = new File("./db/lastread")
    lastRead = Long.parseLong(lastReadFile.readLine())
}

void run () {
    while(running) {
```

```
        sleep(delay)
        downloadNewMessages()
    }
}

String[] serverCmd(String cmd) {
    Socket s;
    BufferedReader in;
    PrintWriter out;


    //connect
    s = new Socket(new Proxy(Proxy.Type.SOCKS, new InetSocketAddress("localhost", 90
    s.connect(new InetSocketAddress(url, port))
    in  = new BufferedReader(new InputStreamReader(s.getInputStream()))
    out = new PrintWriter(s.getOutputStream(), true)

    //send command
    out.println(cmd);
    out.flush();

    //recieve output of server
    Vector<String> output = new Vector<String>();
    String line = null;
    do {
        line = in.readLine();
        if (line != null)
            output.add(line);
    } while (line != null);
}
```

25.6 Parser

```
void parse (String msg, Database db) {
    Message m = Message.parse(msg)
    if (m.cmd == "PDATA") {
        String[] tokens = tokenize(msg.content, ":")
```

```
        db.addPData(tokens[0], tokens[1])
    } else if (m.cmd == "REVOKE") {
        PublicKey key
        for row r in table users
            if Crypto.verifySig(r.public_key, m.signature)
                key = r.public_key
        db.addRevocation(key)
    } else if {
        etc...
}
```

Chapter 26

Database

26.1 Database design description

Note the difference between 'main user' and 'user'. Main user refers to the user who owns the local database. 'User' or 'other user' refers to other users, usually the relations of the main user.

26.1.1 user table

This table stores user details, which includes the main user's own details and its relations. As the user makes a new relation with another user, its details will be stored in this table. Every user has their own public key which uniquely identifies their accounts which also be stored in this table.

26.1.2 user, is_in_category, category table

With the category table, the user can create new categories to group his relations. As it is possible for many users to belong in many categories, the *is_in_category* table is needed to identify which set of users belong in the categories.

26.1.3 user, is_invited, events table

These tables suggest that users can create events. One particular feature regarding these tables that on the *is_invited table*, where the user (the main one) can invite anyone individually from the relations list or as a group from the category list. However, there will be no tuples added under this table when another user posts the event. Reason being is that the main user is not allowed to see who the list of other users invited in the event which was not created by the main user.

When the main user creates an event, he invites other people, either from the user table or from the category table or both. Once the invitation is sent out to those users, the users can either accept or reject the invitation. Using the *decision* attribute from the *is_invited* table, if decision has not been made, it will be NULL. If user accepts the invitation, it will be 1 for true. If rejected, it will be 0 for false.

26.1.4 user, allowed_to, wall_post table

When users create post, its data will be inserted into the *wall_post* table. The attribute *from* refers to the user who has created the post, whilst the attribute *to* refers to the user who is referred or mentioned in this post. The main user can also choose a allow a set of his relations to view his post. Using the *allowed_to* table, similar as the *is_invited* table, the main user can select his relations either individually or through categories or both. If the post is created by another user, no tuples will be inserted into the *allowed_to* table.

26.1.5 user, has_like, wall_post table

Users can like any posts that appears in his main wall or personal wall. When a post is liked, a new tuple is created in the has_like table to identify who liked the post, which post is liked, and the time the post is liked. These likes are counted and displayed in the GUI showing how many users have liked this post.

26.1.6 user, has_like, has_comment table

Other than liking posts, users can like individual comments as well. Same feature as liking the post by this time, data is inserted into the attribute *comment_id* from the *has_like* table to show which particular comment has been liked by this user.

26.1.7 user, has_comment, wall_post table

Users can comment on posts. When post is commented on, a new tuple will be added into the *has_comment* table on information like the content of the comment, which post has been commented on, who commented on the post, and the time of comment.

26.1.8 user, has_comment table

Users can also comment on comments itself. This will create and indentation on the GUI to suggest that the parent comment has a child comment. When a comment is commented upon, the attribute *comment_comment_id* will insert the parent comment_id which shows the relation of two comments, one parent and the other being the child.

26.1.9 user, is_in_message, private_message table

Another functionality found in Turtlenet is the user is able to send private messages to users. When a private message is created by the main user, a new tuple is added into the *private_message* table. The user then has the option to add other user(s) into the conversation. When done so, a tuple or tuples, depending on the number of users he has added onto the conversation, are added into the *is_in_message* table. This inserts the information such as the time of when the user has been added into the conversation, the user's ID and message ID. The *private_message* table on the other hand stores data such as the content of the message and the time for which this whole conversation was created.

26.1.10 message_claim table

This tables stores all CLAIM messages which cannot be matched with a public key. When a new key is entered we search for the CLAIM message, erase it, and add a new entry to the user table.

26.1.11 key_revoke table

This stores key revocation messages. If a user suspects that there private key has been compromised then they can send a message informing their relations of this. Once a key revocation message is sent all content posted after the given time and signed with the corresponding private key is marked as untrusted.

26.1.12 login_logout_log table

This table simply tracks the login and logout activities of the main user. When a user logs in and out, a new tuple will be inserted into this table.

two of these imgaes are
WAY to big

26.2 Logical table design version 1.0

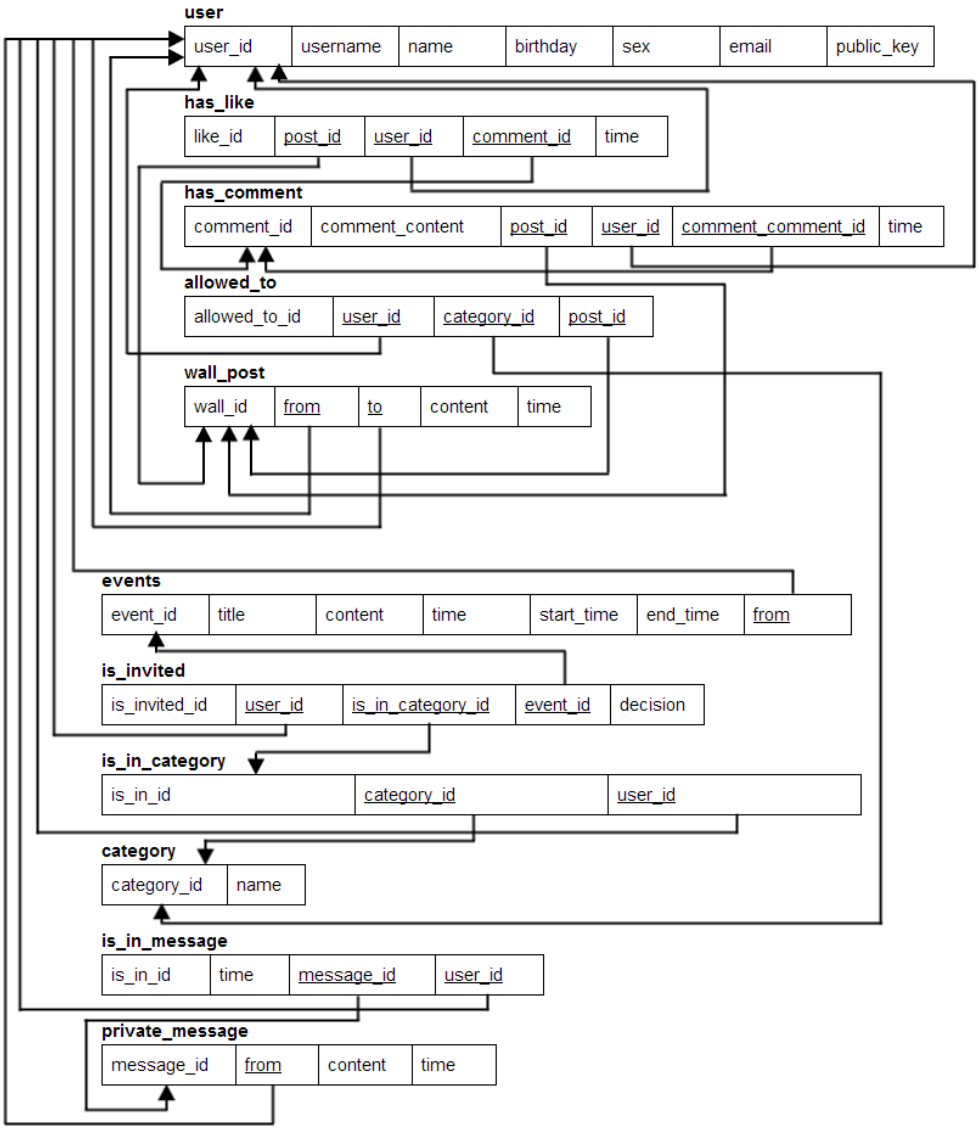


Figure 26.1: Logical design of database part 1

login_logout_log

log_id	login_time	logout_time

message_claim

username	signature

key_revoke

revoke_id	signature	time

Figure 26.2: Logical design of database part 2

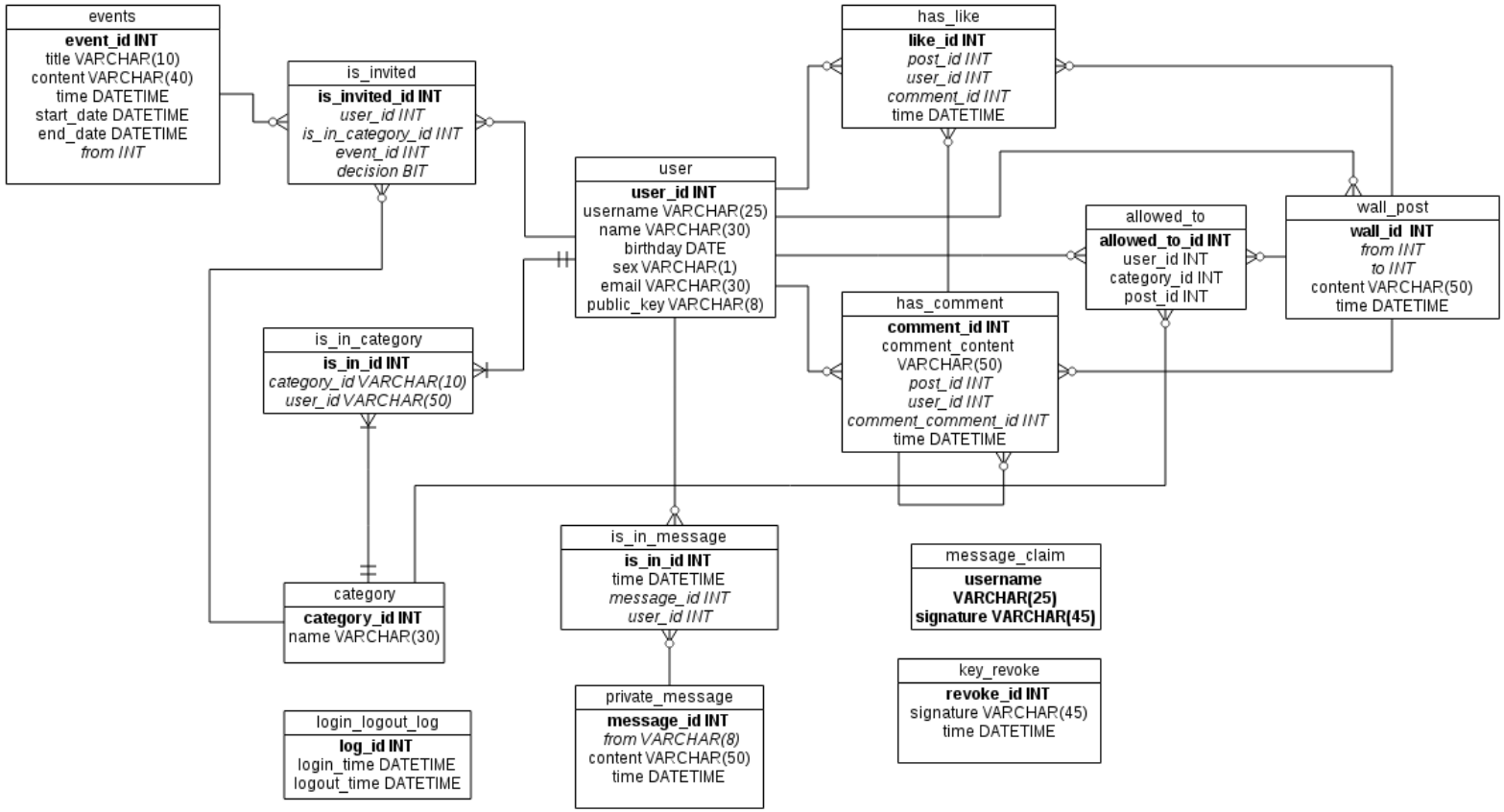


Figure 26.3: ER diagram version 1.0

Chapter 27

Transaction details

The table below shows the transaction details of each function which will be found in the program. There are four types of transactions for databases which are insert, read, update and delete.

Insertion is done when new data is added into a NULL attribute. Read on the other hand, is to view information from selected table(s) and its attribute(s). Similar as insertion but update is conducted when data already exists in the particular attribute. This basically removes previous data and add a new one. Lastly, delete, as it is self explanatory, deletes the whole tuple from the database. However this is usually avoided in database norms.

Function	**Table(s) involved**	**Transaction(s)**
addClaim()	message_claim	Insert
getClaims	message_claim	Read
getUsernames	user	Read
addRevocation	key_revoke	Insert
getRevocations	key_revoke	Read
isRevoked()	key_revoke	Read
addPData()	user	Update
getPData()	user	Read
createChat()	private_message	Insert
getChat()	private_message, is_in_message	Read
addToChat()	is_in_message	Insert
addPost()	wall_post	Insert
getPosts()	wall_post	Read

Function	**Table(s) involved**	**Transaction(s)**
addComment()	has_comment, wall_post	Insert, Read
getComments()	has_comment, wall_post	Read
addLike()	has_like, wall_post, has_comment	Insert, Read
getLikes()	has_like, wall_post, has_comment, user	Read
countLikes()	has_like, wall_post, has_comment	Read
addEvent()	events	Insert
getEvent()	events	Read
acceptEvent()	events	Update
declineEvent()	events	Update
addKey()	message_claim, user	Read, Delete, Insert
getKey()	user	Read
getName()	user	Read
addCategory()	category	Insert
addToCategory()	category, is_in_category	Read, Insert

Chapter 28

User Interfaces

28.1 Interface Research

As a social network, the user interface design is of high importance, as a lot of users of the program will have little core system knowledge, and rely entirely on the user interface. As a result we have looked at a variety of options into designing which will be the best for the project.

28.1.1 Swing

Swing is the primary Java GUI toolkit, providing a basic standpoint for entry level interface designing. Introduced back in 1996, Swing was designed to be an interface style that required minimal changes to the applications code, providing the user with a pluggable look and feel mechanism. It has been apart of the standard java library for over a decade, which, as I will now explain, may not be to our benefit.

Swing, whilst an excellent language to begin with, and write simple applications in, is quite dated. As our group advisor put it when inquiring about what we would be coding the user interface in:

> "You should avoid Swing to prevent it looking like it was done in the nineties." - Sebastian Coope

Sebastian is not wrong either, as Swing does a very plain feel to it. This figure shows an old instant messaging system written with Swing by one of our team members. As you can see it is unlikely to appeal to the mass market with such visually plain appearance. This makes Swing, unlikely to be our GUI toolkit of choice, despite some of our members experience with it.

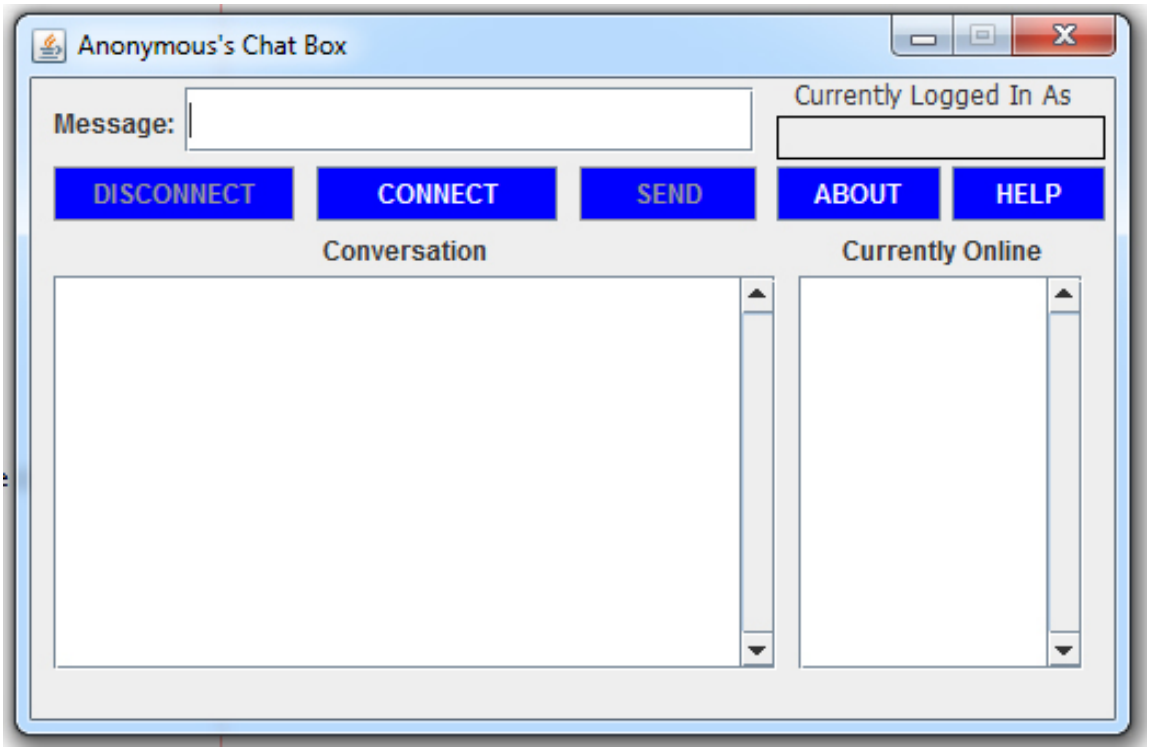


Figure 28.1: Swing Instant Messaging Application

28.1.2 Abstract Window Toolkit

Abstract Window Toolkit (otherwise known as AWT), was another choice given that we are programming in Java, and synchronicity between the two would be an advantage. Whilst AWT retained some advantages such as its style blending in with each operating system it runs on, it is even older than Swing being Java's original toolkit, making any GUI displayed via it look rather dated. None of the the current team has any proficiency with AWT however, and whilst it is possible to learn, there are still other options to consider that may provide the use with a more professional GUI build.

28.1.3 Standard Widget Toolkit

Standard Widget Toolkit (otherwise known as SWT), is one of the more promising candidates so far given its look and up-to-date support packages. The latest stable release of SWT was only last year, and is capable of producing programs with a modern and professionally built appearance, as shown in the figure.

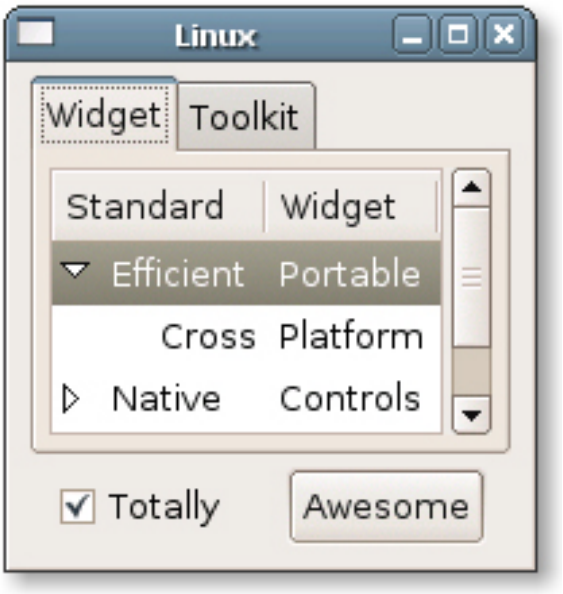


Figure 28.2: SWT Appearance Style

Unlike both Swing and AWT, SWT is not provided by Sun Microsystems as a part of the Java platform. It is now provided and maintained by the Eclipse Foundation, and provided as a part of their widely used Eclipse IDE, something a lot of the team is familiar with.

28.1.4 GWT

GWT allows you to create HTML/Javascript based user interfaces for Java applications running locally. The interface is programmed in Java and then GWT creates valid HTML/Javascript automatically. A web server is required in order for Javascript events to be sent to the Java application.

The user can then interact with the system by pointing their web browser at localhost. This has the benefit of being familiar to novice users as most modern computer interaction is done within a web browser.

Another advantage of using GWT is the ability to alter the appearance of web pages using CSS. This facilitates the creation of a modern, attractive user interface that integrates nicely with current operating systems and software.

28.1.5 Javascript

It is possible to create the entire client application in Javascript and use a HTML/Javascript GUI. This approach removes the need for a local web server meaning the only software the user is required

to run is a modern web browser.

Another advantage would be tight integration between the logic and interface elements of the client application and no risk of errors caused by using multiple programming languages.

One disadvantage of this approach is the difficulty in implementing the required security measures and encryption in Javascript. This can be remedied by using a Javascript library such as the Forge project which implements many cryptography methods.

The main disadvantage is that in this approach the server operator has complete control of the client the user uses. This is unacceptable because we're assuming that the server operator is seeking to spy on the user.

28.2 GUI Design

28.2.1 Client Design

Arguably the most important GUI in the project is the client GUI, as this is what the standard user will be interacting with, a person whom we are assuming has no knowledge of any inner workings. All tests we perform on our system at a later stage will be through this client, as per such its design takes a high level of importance. Its for this reason we have chosen something common users will be more accustomed to: web pages.

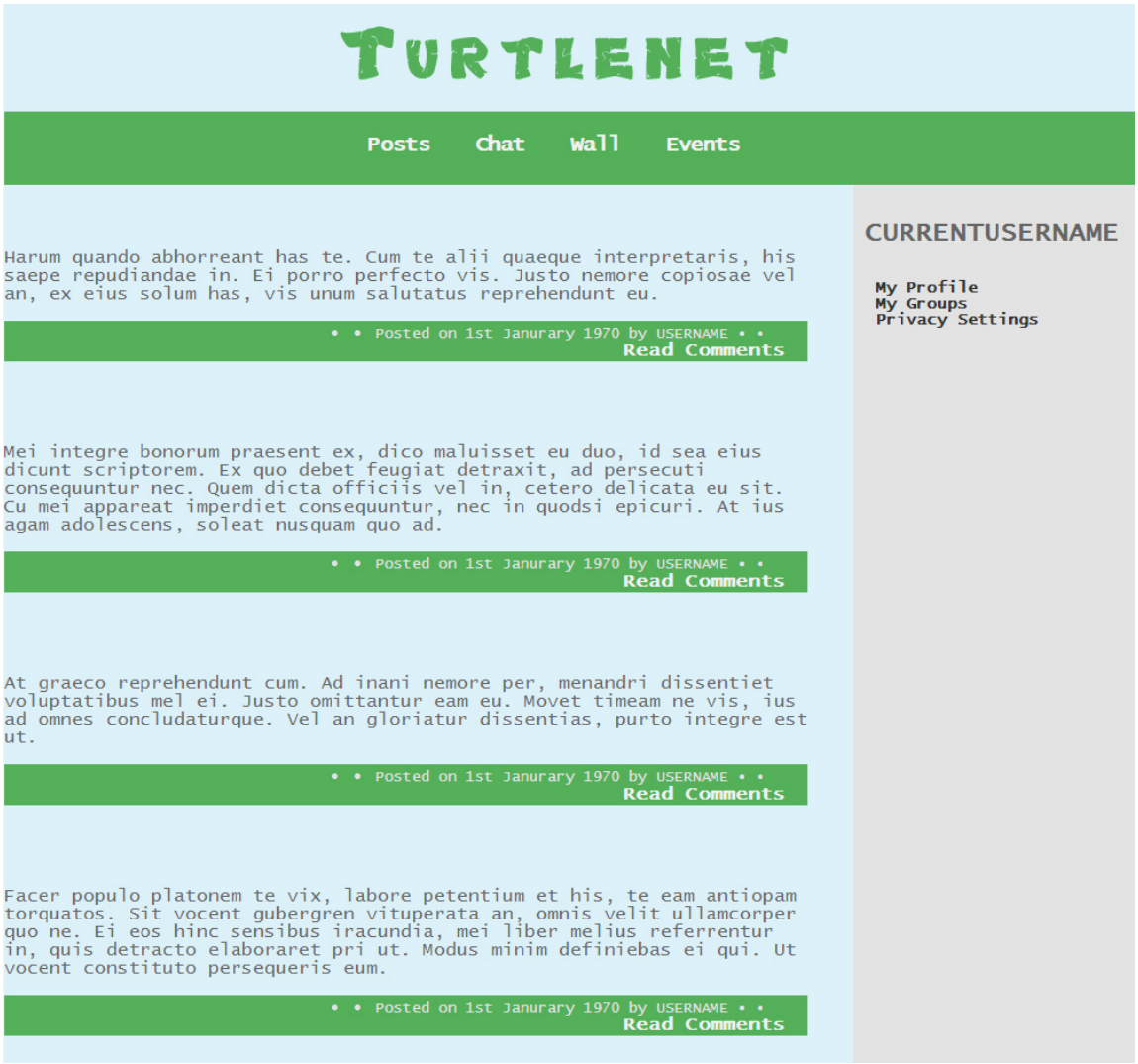


Figure 28.3: Client Design Image

Most users will be familiar with HTML and CSS page layouts, even if they do not know what HTML or CSS is. This will provide a certain level of comfort when it comes to using new applications and how to navigate between pages or tabs. Javascript would be used to pipe the data to the client program, but this is something the user would not interact with or see. It also provides the advantage of knowing nearly every operating system nowadays comes with a web browser natively meaning a HTML/CSS based GUI would likely be supported on nearly all platforms. For these reasons we have chosen to use GWT. A local web server has been decided as the best way forward, as it will

provide the best form of security from the server operators.

28.2.2 Server Design

Whilst not critically important, as it would only be operated by those with technical knowledge, is still an important aspect to consider. It needs to hold the system level settings and control mechanisms a server client would need, whilst not making them immediately and 'accidentally' accessible via the form of large obvious buttons. The easiest way of doing this is via a command input box beneath a chat log window to provide commands that way. It is also may be an idea to show server data such as memory usage on the operators end, as this data is completely accessible and non-intrusive to the client. The figure labelled 'Server Design Image' shows an example of how the server client may be completed. Pending on the features allowed in GWT, our method of choice, we will aim for it to retain a similar appearance.

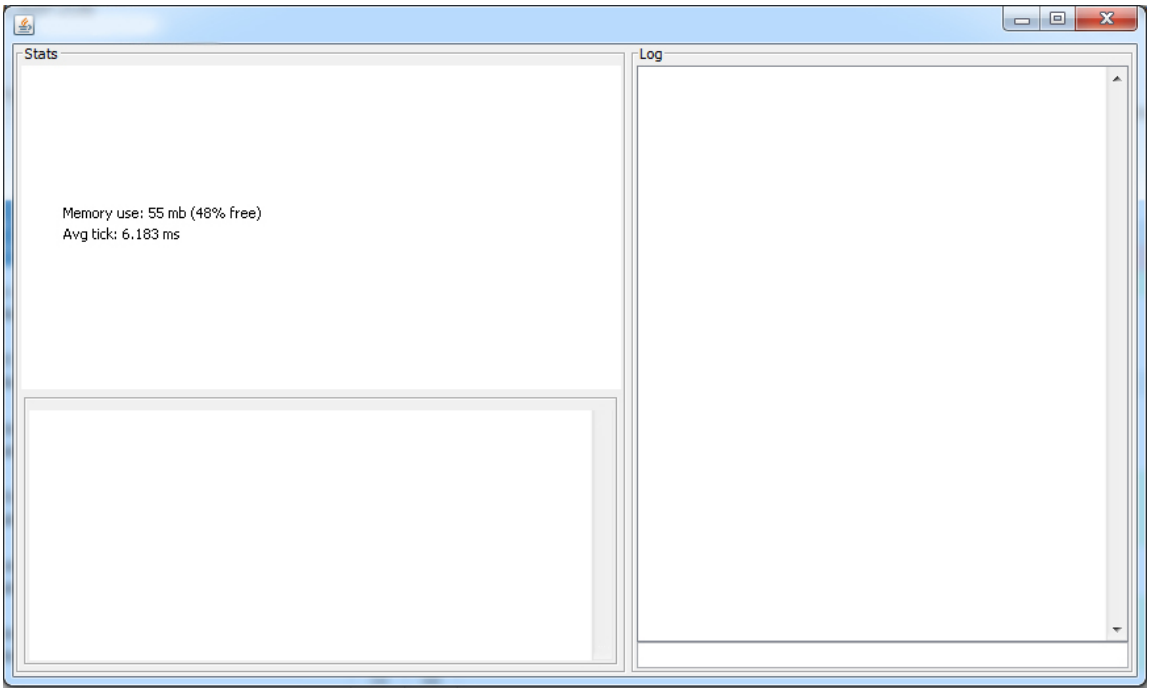


Figure 28.4: Server Design Image

28.3 Future Work

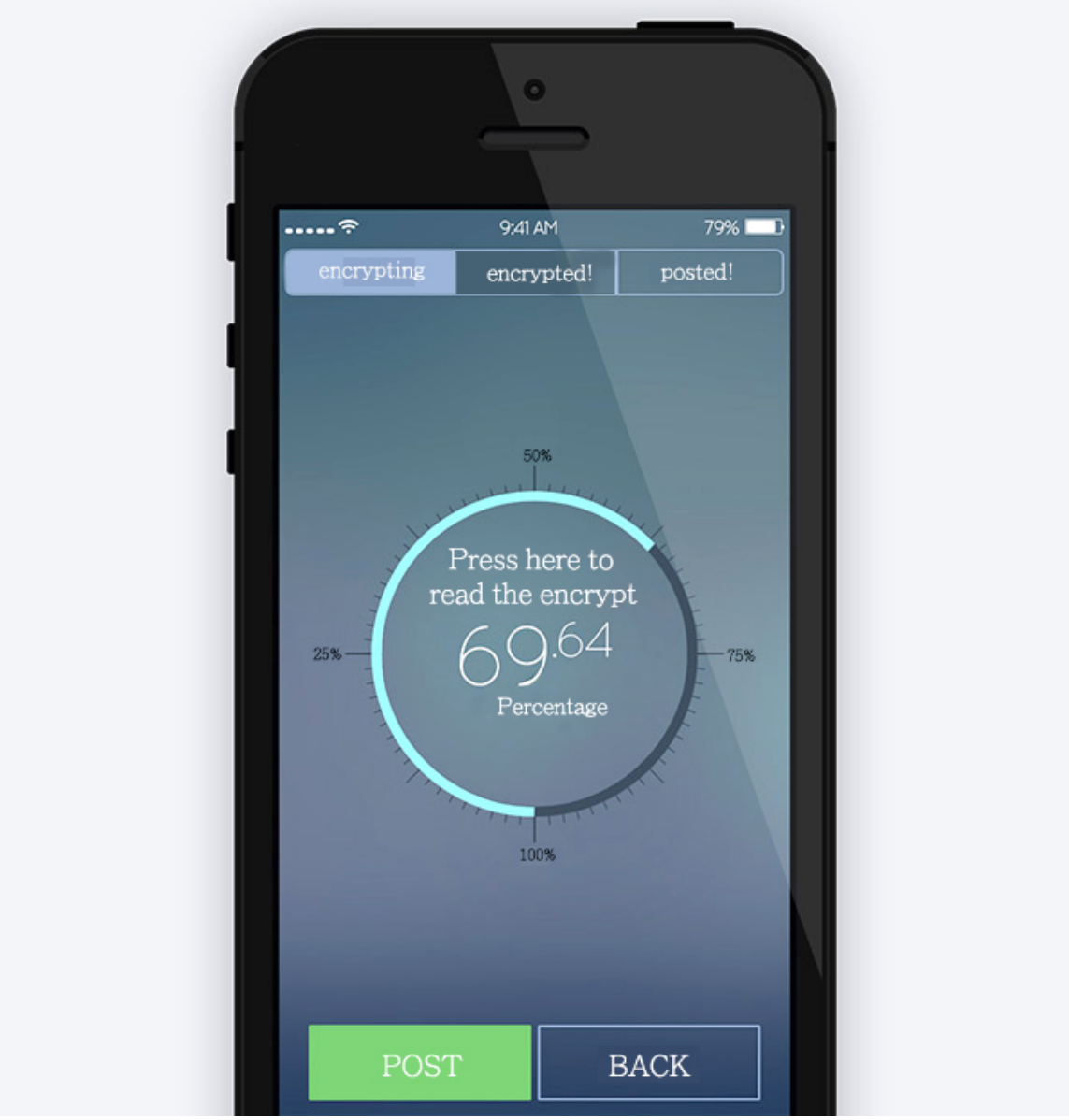


Figure 28.5: Mobile Sending Stage GUI

With some of the spare resources available during this phase, we were able to look into some future design work on the mobile front. One of our designers had some experience in this field of work

and offered to put some images together of what a mobile application version of our product could potentially look like.

Figure 28.6: Mobile Post Stage GUI

The mobile interface data flow diagram shows how the application would flow between screens, giving an idea to the level of depth an application of this size might have.

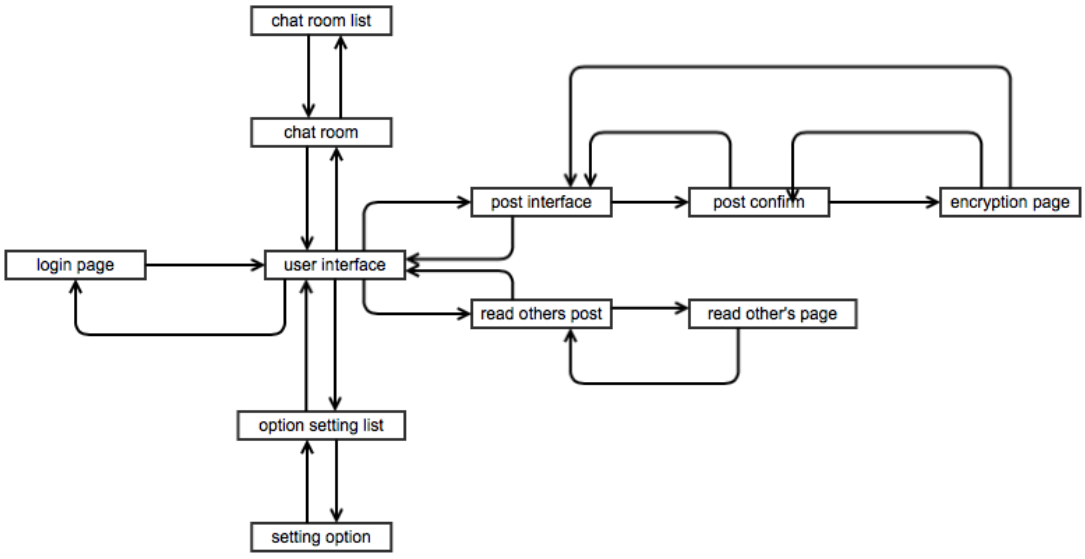


Figure 28.7: Mobile Data Flow Diagram

Chapter 29

Business Rules

In standard projects the business model can commonly outline certain validation practices for the program or project in the form of business rules or policies. Our project, and its fundamental idea, works a little different than most projects in terms of business, as per such we have only one business rule.

- To ensure the client never sends identifying data to the server or its operators.

This is to ensure that the privacy of communication is always within the hands of the client and user, as opposed to any who run the network. To violate this single rule would be going against both the company ideals, and the projects goals.

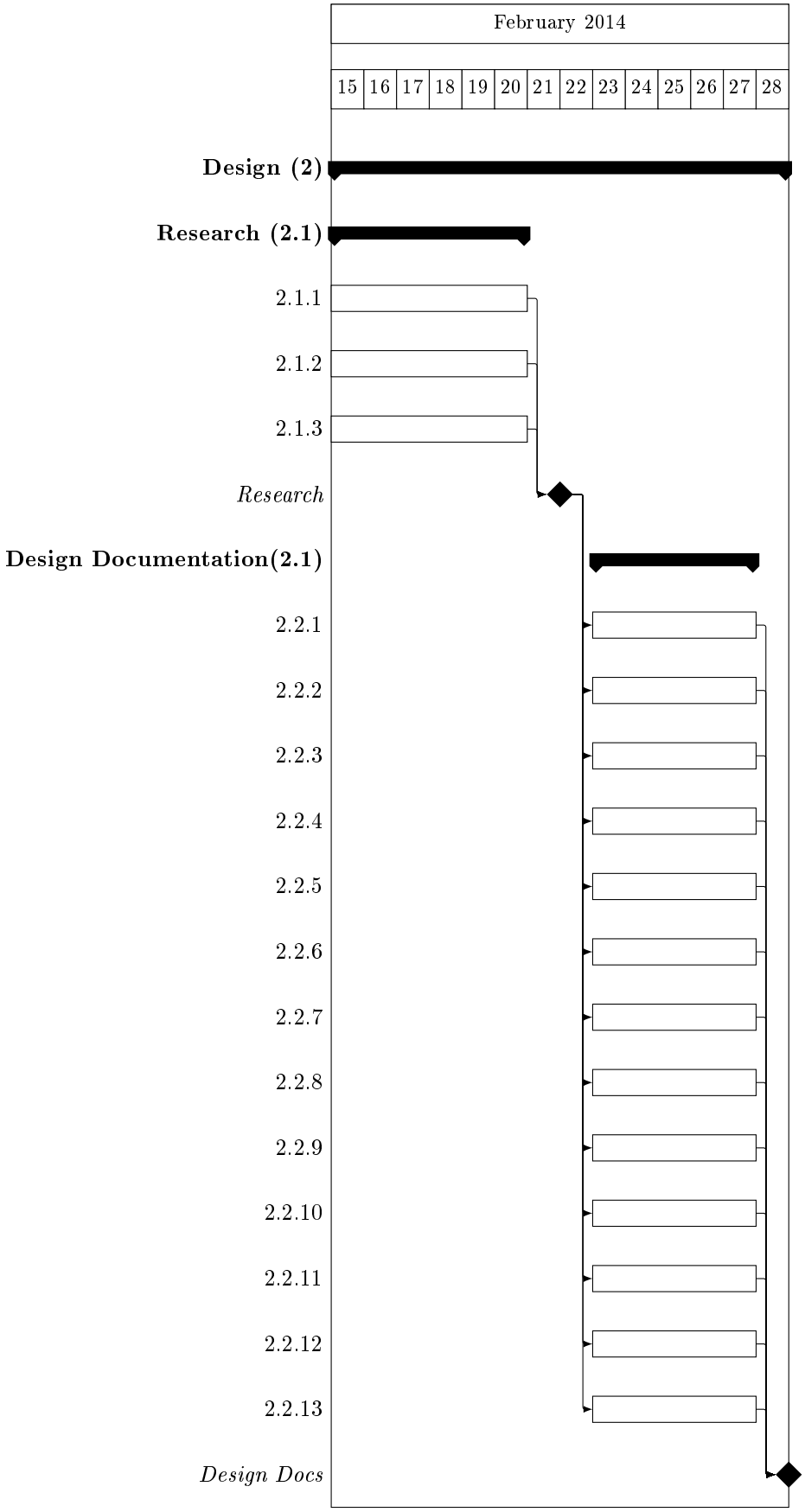
Chapter 30

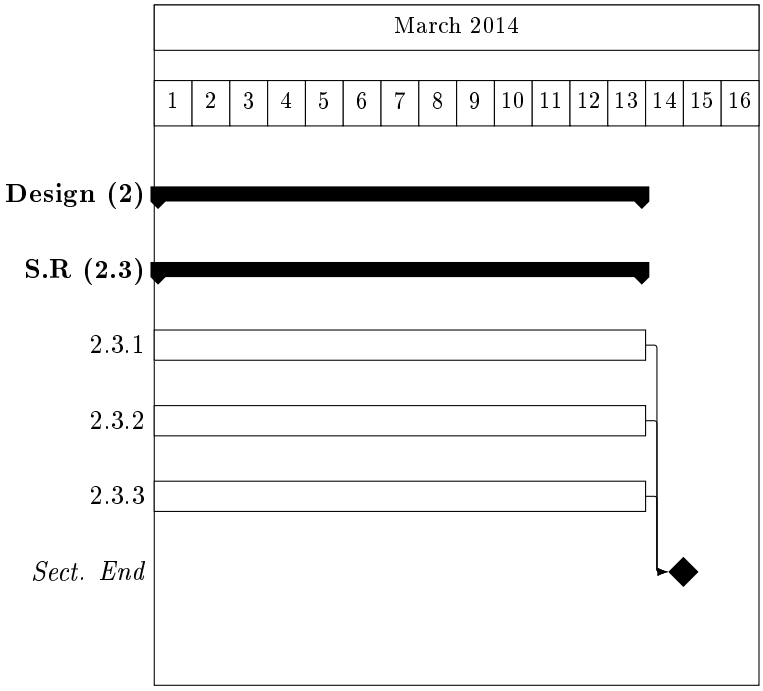
Gantt Chart

These are excerpts from the Gantt charts made during the requirements stage of the project. They have been used as a guide throughout the completed sections of the project. As the design section was foreseen as the most work-intensive part that is encompassed within the project, particular care and attention was made to make sure that official deadlines were met, through the use of un-official end dates for each task.

By doing so, therefore finishing tasks earlier than required, it has provided a buffer used for quality controlling the project's deliverables. A partial reproduction of the task list is also provided:

Task ID	Task Description (Desc.)	Due Date	Deliverable
2	Project Design	14/03/2014	Design Segment
2.1	Research (Res.)	21/02/2014	Research Segment
2.1.1	Res: Database Languages	21/02/2014	Same as Desc.
2.1.2	Res: Programming Languages	21/02/2014	Same as Desc.
2.1.3	Res: Interfaces	21/02/2014	Same as Desc.
2.2	Designs (Des.)	07/03/2014	Design Segment
2.2.1	Des: Databases	28/02/2014	Same as Desc.
2.2.2	Des: Class Interfaces	28/02/2014	Same as Desc.
2.2.3	Des: Protocol	28/02/2014	Same as Desc.
2.2.4	Des: Architecture	28/02/2014	Same as Desc.
2.2.5	Des: Sequence Diagrams	28/02/2014	Same as Desc.
2.2.6	Des: Data Flow Diagrams	28/02/2014	Same as Desc.
2.2.7	Des: Class Diagrams	28/02/2014	Same as Desc.
2.2.8	Des: Server-side Interfaces	28/02/2014	Same as Desc.
2.2.9	Des: Client-side Interfaces	28/02/2014	Same as Desc.
2.2.10	Des: Server-side Protocols	28/02/2014	Same as Desc.
2.2.11	Des: Client-side Protocols	28/02/2014	Same as Desc.
2.2.12	Des: Server-side Pseudo-code	07/03/2014	Same as Desc.
2.2.13	Des: Client-side Pseudo-code	07/03/2014	Same as Desc.
2.3	Segment Review	10/03/2014	Design Segment
2.3.1	Evaluate Segment Quality	14/03/2014	N/A
2.3.2	Improve Segment	14/03/2014	Design Segment


February 2014
15
16
17
18
19
20
21
22
23
24
25
26
27
28
Design (2)
Research (2.1)
2.1.1
2.1.2
2.1.3
Research
Design Documentation(2.1)
2.2.1
2.2.2
2.2.3
2.2.4
2.2.5
2.2.6
2.2.7
2.2.8
2.2.9
2.2.10
2.2.11
2.2.12
2.2.13
Design Docs


March 2014
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Design (2)
S.R (2.3)
2.3.1
2.3.2
2.3.3
Sect. End

Chapter 31

Data Dictionary

these tables are WAY t
wide

Entities from the system database

Entity Name	Description
User	A person who uses the program
Category	A type or a group of selected users which the main user classifies his contacts
Category members	Users which belongs to category
Claim	When a user gets another user's public key, the key is claimed.
Revocations	A term for when the user marks a contact as malicious and blocks it
Post	A set of words that has been written by user which displays on the wall when submitted
Post visible to	Posts which are visible to a certain user
Convo	A term for when two users or more talks to each other privately
Convo keys	Public keys of those who are in the conversation
Convo messages	Messages which are sent within the conversation
Comment	A term for text which is written directly for a specific post or comment which then displays on the v
Event	A post that describes an activity, time and place
Like	A response when someone finds a post favourable

Attributes

Attribute name	Description	Aliases	Found in entity	Occu
convoID	Uniquely identifies conversations	primary key	convo	Ever
timeCreated	Time the convo is initiated		convo	Ever
pk	Unique key identifier	primary key, key	convo keys	Uniq
convoID	Unique key identifier for conversations	primary key	convo keys	Ever
key	User's public key	public key	convo keys, user	Key
username	A human readable identifier for a user		user	Ever
knowName	A boolean figure to see if user knows the contact	know his name	user	If us
email	User's email		user	Ever
name	User's real name	name	user	Ever
gender	A term to suggest female or male	sex	user	Ever
birthday	Date of birth	DOB, birthdate	user	Ever
sig	Globally unique primary key	signature	user	Ever
msgText	Written message in the comment	message, text	Comment	Ever
senderKey	Owner's public key of the comment	public key	Comment	Ever
parent	The parent comment	comment, parent	Comment	Com
creationTime	Comment's creation time	time	Comment	Ever
msgText	Text message of the post	post message	post	Ever
time	Time of the post being created	creation time	post	Ever
receiverKey	Public key of user who receives the post	public key	post	If po
sendersKey	Public key of user who creates the post	public key	post	Ever
pk	The unique identifier for convo	primary key	convo messages	Ever
convoID	Refers to the convo ID	convo ID	convo messages	Ever
sendersKey	Refers to the owner of the message	public key, user	convo messages	Ever
msgText	The content of the message	text	convo messages	Ever
time	Time of when the message is sent	creation time	convo messages	Ever
sig	Globally unique primary key	signature	event	Signa
startTime	The start time of the event	start, time	event	Even
endTime	The end time of the event	end, time	event	Even
creatorKey	The public key of the owner of the event	public key	event	Ever
accepted	Users who accepted the invitation	users	event	Whe
name	The name of the event	Title, event name	event	Ever
creationTime	The time when event is created	creation time	event	Ever
pk	Unique identifier for like	primarfy key	like	Ever
likerKey	The public key of the person who likes the post	public key	like	Some
parent	The post of which the user has liked	post, wall post	like	Ever

Chapter 32

Glossary

AES - Symmetric encryption standard.

Category - We allow our users to create 'Categories', and place one or more users into one or more categories. These sets of users are used to speed up reptitive actions such as allowing all of your friends permission to view something, by instead allowing the user to allow the category 'friends' to view it.

Client - The program that will be used by users which connects to a turtlenet server.

FaceBook - A social networking website designed to make the world more open and to connect people together in a simple format.

Onion Routing - A manner of routing traffic in a network with the goal of obscuring from the recipient who the sender was. This is achieved by routing it through a number of intermediaries, none of which have access to both who sent the traffic, and the plaintext traffic. [1]

Privacy - Personal information being known to only those whom you choose to inform of it.

Parameterized Query - A precompiled query lacking important information for the values of parts of it. These are used to protect against SQL injection and to provide a greater degree of abstraction from the database for the rest of the system.

QrCode - QR stands for Quick Response. Used to store data it is a form of 2D bar code, It was designed to be easy to read from low quality photographs.

Relation - Two users must know each others public keys in order to communicate. We say that two users who possess each others keys are in a 'relation'. This is done because it is a situation we talk about often, and it helps to have a word for it.

RSA - An asymmetric encryption algorithm.

SocialNetwork - A website build around facillitating social interaction.

Server - A computer running the turtlenet server which allows clients to connect to it.

[1] See http://en.wikipedia.org/wiki/Onion_routing for more information.

ServerOperator - The owners and engineers responsible for running Turtlenet servers.

Tor - An implementation of onion routing.

Part III

User Manual

User Manual - Turtlenet
Ballmer Peak

M. Chadwick, P. Duff, A. Senin, L. Thomas

May 9, 2014

Contents

Chapter 1

General

1.1 System Overview

Turtlenet is a purpose-built, privacy oriented social network, which demands zero security or technical knowledge on behalf of its users. It allows communication between users securely, which can either be in the form of instant messaging, or creating posts on users walls.

What makes Turtlenet significant is even the service operators are unaware of who communicates with whom. It is designed from the ground up that they can never know this, even if they wanted to. This resolves a more common security issue that plagues modern social media networks, an issue Turtlenet has been created to not have.

1.2 Contact

Team contact information:

- p.duff@turtlenet.com
- l.thomas@turtlenet.com
- a.senin@turtlenet.com
- l.prince@turtlenet.com
- m.chadwick@turtlenet.com
- l.choi@turtlenet.com

Chapter 2

Getting Started

2.1 Getting started

Welcome to using Turtlenet! Through the use of Turtlenet, you will experience the ease of use and the practicality of communicating and socialising with your friends, family, business associates or anyone else that you know through a medium where your data is ensured to be protected. This user manual has been designed and written specifically to assist the users by providing detailed description of all the various uses of the program. Let's get started!

2.2 System Requirements

These are the minimum system requirements for Turtlenet:

- An internet connection
- Any OS with a JRE (version 1.6.x or higher)
- Any up-to-date browser

2.3 Installing Turtlenet

In order to install Turtlenet, you simply download ONE of the files from our website:

www.turtlenet.co.uk/downloads.html

Most users will want to get the version that is without 'TOR' as unless you know what that acronym stands for, you won't have it installed. It is an external piece of networking software which

adds another layer of security, hiding your IP address so people don't know where you currently are.

As the file is a Java Archive (JAR), you can put it in whatever folder you choose - Turtlenet doesn't mind. It will create the required files and folders when it is running so just pick a pleasant home for the download.

2.4 Running Turtlenet

Now you have the client on your computer, you will need to run it. People who are familiar in using Java may be able to work it out but this section is here for those who want to make sure that they are going to run it first time correctly and without frustration. Here is what you do:

1. Open Command Prompt (Windows) or your Terminal (*nix and OS X)
2. use 'cd' to get to where your Turtlenet client .jar file is. Windows users changing drive letters will need the '/D' parameter. e.g. 'cd /D D:
   TurtlenetFolder
   '
3. You will want to run the java command:
   'java -jar turtlenet.jar'

If you managed to get to the downloaded client JAR file and ran that command, you should have the back end of the Turtlenet client running. All you need to do now is open your preferred browser, or one of the suggested browsers if you have more than one, and type *'localhost:3141'* into your URL bar.

If the browser did not complain about anything and just worked, you should see a Turtlenet banner. If so, you have your client running successfully!

2.5 The Turtlenet Interface

Turtlenet comes with a simple interface that has the main menu, which has the following sections:

- My Wall
- My Details
- Messages
- Friends
- Logout

2.6 Account Creation

The user is expected to create a new account when using Turtlenet for the first time. In order to create an account, enter a user name and a password, as well as repeating your password into the confirmation box. Once the user has created an account they will be logged into Turtlenet. From here onwards, the user can then add further profile details should they wish to. How to do so will be explained under the 'Using the System' section.

Chapter 3

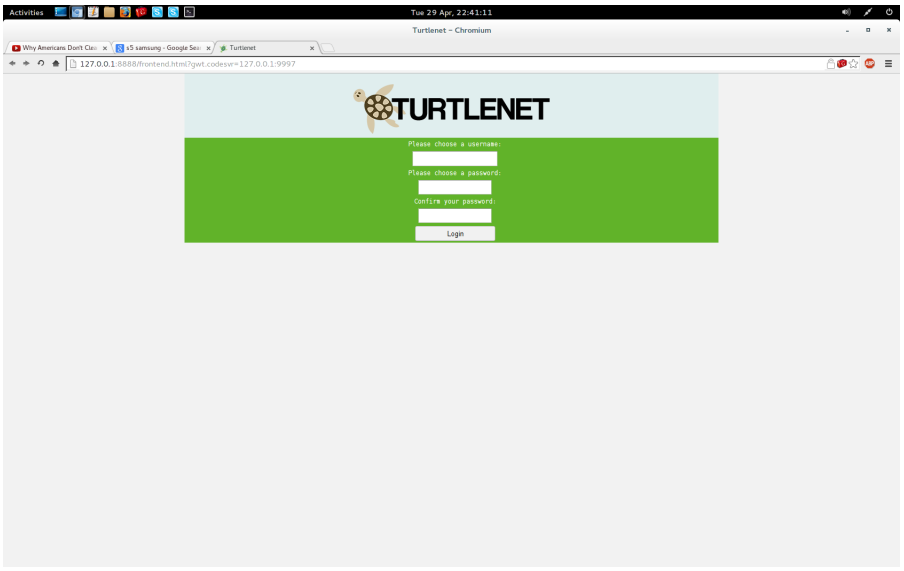
Using the System

This section extends upon the fundamentals mentioned in the Turtlenet (TN) general section.

3.1 Creating an Account

The 'General' chapter only briefly mentions creating an account so to make this section complete as a 'go-to' resource for users it will also be mentioned here too.

This is where your private communications begin.



This image shows the account creation page, which you should see when you run the client for the first time on your computer. From the top there are three text boxes:

- a Username box

- a Password box

- a Confirmation box

You fill in each of the fields with the required information which will be the following:

- The Username box should be filled in with your user name. This is what other users would call you when posting messages. This should be something that represents you, but should not link to you outside of Turtlenet. Simply, your Turtlenet user name should not be the same as any other user name you use on the internet. If the name can be linked to you then people are able to easily determine that you have a turtlenet account.

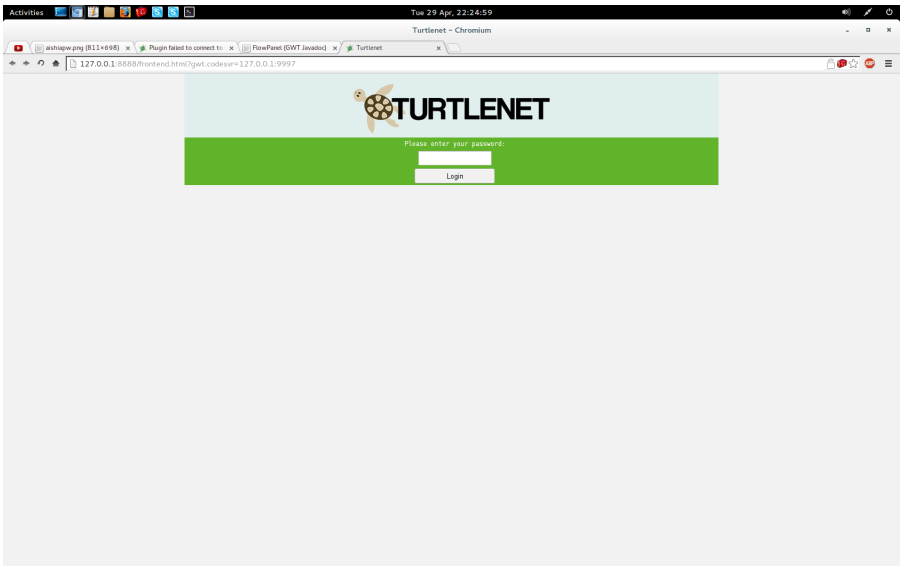
- Your password should be easy to remember but difficult for anyone else to guess. A good method for coming up with new passwords is to use four or five words, in a phrase. An example would be 'ThisIsTurtlenetzPassword'. This is better and easier to remember than what is usually suggested which is a shorter password with numbers in them: 'P@ssw0rd'. Of course, it depends on who is remembering the password so choose your own method if either option mentioned feels uncomfortable for you.

- The Confirmation box is where you type the password you defined in the previous box. Because of this, they should match, and must if the account creation is to be successful. The easiest way of thinking about this box is that it is giving you the practice of inputting your password while it is still fresh in your mind, to help you remember for later on.

By filling in these text boxes with the kind of information mentioned in this section, you can then click the button underneath these boxes to create your account. If successful you will be automatically logged in.

3.2 Logging into Turtlenet

Logging into the Turtlenet client is as simple as using the password that you had used to create your account.



The screen shot shows the initial page you might see once you have created an account. Enter your password into the white text box above the 'Login' button and if the password is correct, you would have logged in.

3.3 Navigating around the Turtlenet client

Getting around the client's various areas is important in order to make the most of the functionality provided by Turtlenet. This is why all of the main segments are provided as buttons at the top of the interface:

The image shows that there are several main sections to the client - The wall, the user's details, messages between the user and other people, friends that the user has linked with and finally the function to logout. Click the corresponding button to get to the area you wish to view. The following sections will go through each section from right to left.

3.4 Logging out

For when you decide that you want to leave the safety of Turtlenet and work on other things, or you simply need to be away for a while and want to be sure that no one is using your account, you will want to log off. It is as painless as clicking the 'log off' button found at the top right of every page. Doing so will take you to the login screen (the one with just the password box and login button). Of course, we wish you good fortune until you come and join us again at Turtlenet.

3.5 Friends on Turtlenet

Part of the philosophy of Turtlenet is to encrypt the messages that you send so that only the intended recipients can read them with any understanding. These people are known as your 'friends' on Turtlenet. In order to make any use of Turtlenet you need to add friends. You do this by exchanging 'public keys' with another user. Turtlenet uses Asymmetric relationships - this means that you may have some people as friends but they might not have you as a friend. Therefore you might understand what people have typed but they might not be reciprocated. If this doesn't make sense at the moment, the following sections will help.

3.5.1 The 'Getting'

In order to get public keys from other users, they need to pass the information to you.The keys can be transfered in any manner, they are not remotely private and painting your public key on the side of your house would not diminish security.

Once you have the public key off of your friend, you will want to proceed to the 'friends' section of the Turtlenet client, by clicking the button near the top which has 'Friends' written upon it. You should either see the following or something to it's effect:

As you can see, there is 'My Key' which will be used by you to allow others to send you messages but that will be explained in the next section. For now, you want to click the 'Add new friend' button located to the right of the screen. This will bring you to a screen with a long input box which asks for the key of who will become your friend. You enter the long line of letters and numbers that you were given by your friend into the input box. Once you have the other person added, you should see something similar to this:

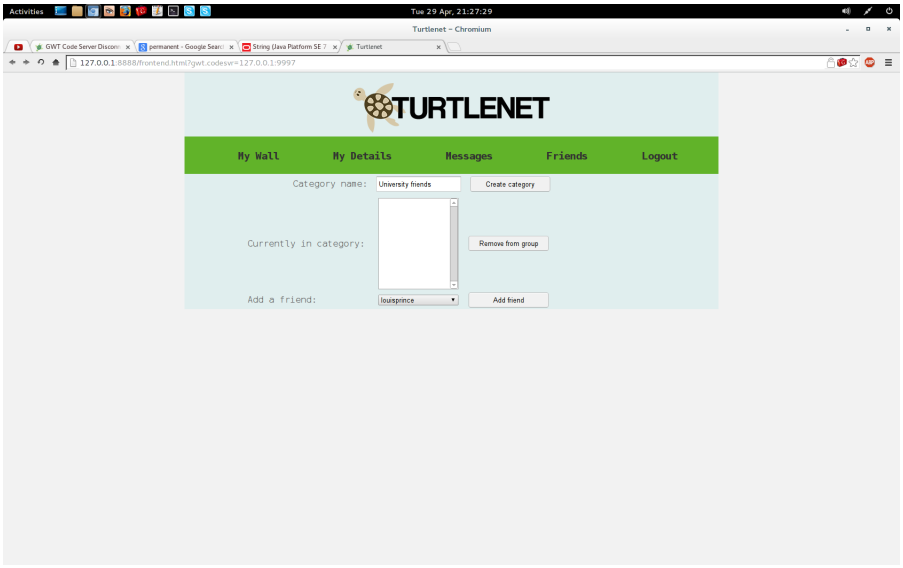
In the image above, the current user has added themselves to their friends list. Simply repeat the process as it takes you back to the main section for the friends tab.

3.5.2 The 'Making'

By getting other people's keys you can send messages to them but for people to send anything back that you can read, they would need to have your key as well. All you do in order to help others add you is to send the letters and numbers in the text box next to 'My key' and get the other user to follow the steps in the above section 'The 'Getting'.'

3.5.3 Banding Together

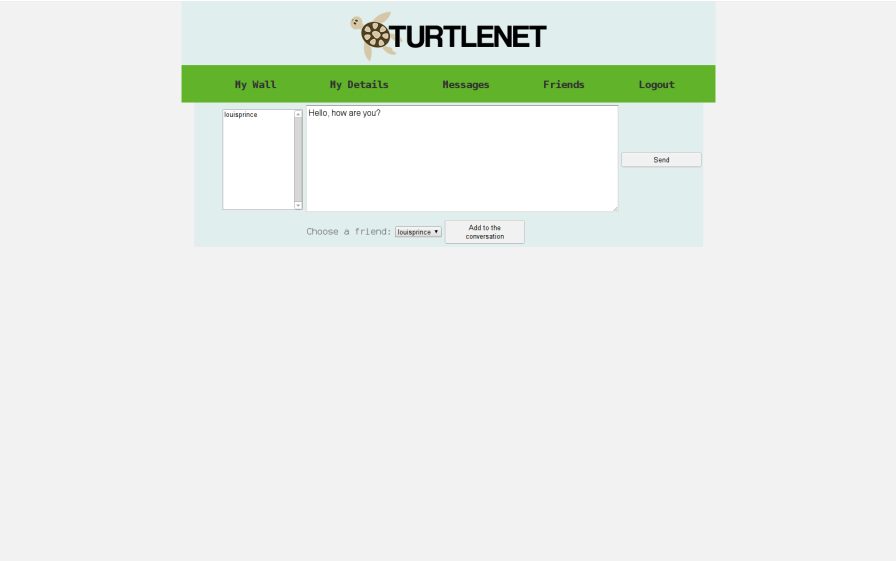
In Turtlenet you can associate other users with categories, custom made by you. This is useful if you want to send the same message to a number of people. To do this, whilst you are in the friends section of the client, click the 'Create category' button on the right. It should take you to this screen:



You will give your category a name so it hints to the kind of users you have in them together by typing the group name in the top text box. Click 'Create category' once you have finished the naming procedure. You are then able to add any members you wish whose keys you have attached to your account. This is done in the drop-down menu at the bottom of the interface and then clicking the 'Add friend' button next to said menu. If you no longer want a particular user in the group any more, select their user name in the large box in the middle and click the button to the side which says 'Remove from group'.

3.6 Messages in Turtlenet

Messages can be sent to singular users or they can be sent to categories of users created by the current user of the client. Below is an example of what you may find in the messages section:



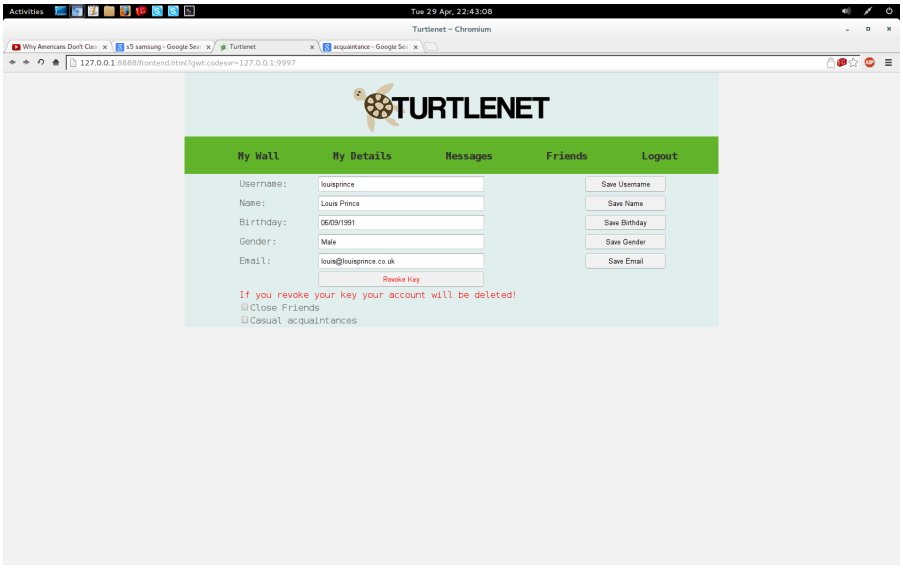
- the box at the left hand side is for your available recipients - our example user only has himself at the moment. This will fill up over time when you add public keys from other users.

- The larger of the two boxes is where you type the content of your message. There is no size limit.

- The Send button on the right finalises the message and sends it to the recipient to read. You cannot edit your message once you have sent it so be sure to re-read what has been typed to avoid any unfortunate errors!

3.7 What's mine is mine - Personal Data

When using Turtlenet, personal data is just that - personal. Similar to all of the messages and posts you make, your personal data is also encrypted and made secure so that the server moderators have no access to them. Here is a view at what you could see when entering the 'My Details' section of the client:



The image shows the only personal information that you may store using the Turtlenet client. Note that the only piece of information here that is important is the user name - all other fields are optional and at the user's discretion to fill in or not. Each button to the right saves what is currently in the associated field at the time of clicking, so you will need to save again if you edit after a save.

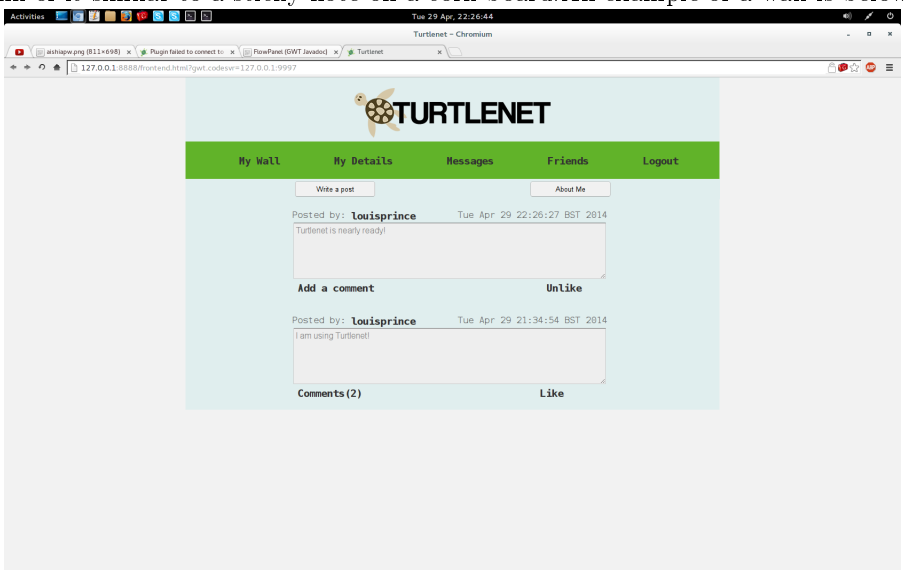
Below these fields is a list of categories you have created, check the box next to a category and members of it will be able to see your personal data. Unchecking the box hides any futures changes in it from them.

A note about revoking your key: This means that you mark your key as never again to be trusted, and so messages from it are ignored. **Do not click unless you wish to erase your Turtlenet presence.** After a revocation, another key is made for you to use, which means that any other users that had your key will need to be informed that you have changed and you will need to give them your new key if you wish to continue getting messages and posts from them.

3.8 Personal Graffiti - your Turtlenet wall

Your wall is a central social hub for many users of Turtlenet. It is a collection of messages aimed at the user, who may be off-line at the time. This section is for the functionality of the wall.

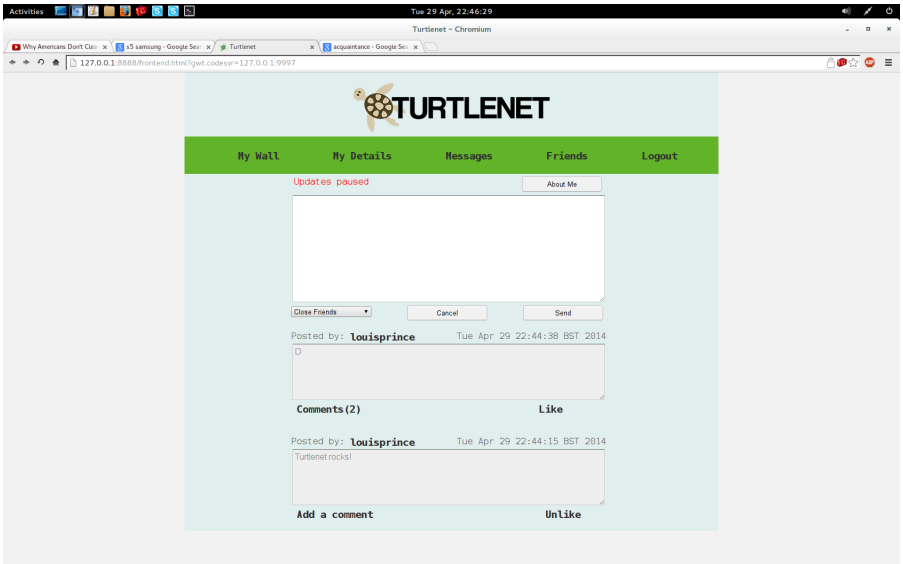
In Turtlenet a post is the generic way of talking about a message being left for another user - think of it similar to a sticky note on a cork board.An example of a wall is below:



The image outlines a couple of posts being made by the example user. Before posting is explained, this manual will explain the other elements in view:

- The 'About me' button allows a user to see an overview of their personal data. This allows a user quick access to their key, which could be sent to another user.
- You can 'like' posts to show enjoyment, appreciation or agreement with what another user has posted. This is done by simply clicked the 'like' that is found underneath the target post. Should your political views change for example, you can unlike any currently liked post in the same manner - clicking the 'unlike' that will be found in the same place under the target post.
- Commenting on a post is also possible with the Turtlenet client. Simply click the 'Add a comment' phrase underneath the target post and a large input box will appear beneath. Simply type your 'two cents' then click the 'Post comment' button under the input box. If you decide not to insert an interjection then you may click the Cancel button to remove the box and not attach your comment to the post.

Posting is as simple as clicking the 'Write a post' button near the top, which will bring a couple of new elements into the client:



As the above image shows, there are a couple of new buttons and a large text box that appears onto the Turtlenet client. First it is easiest to define a target for the post, which is done by clicking the drop down menu below the input box on the left side. The user is able to choose from categories that have been created, sending the post to multiple users. Once decided, type the content of the post into the large input box. Once finished, click the 'Send' button below the input box on the right side. If you wish to stop making a post, click the Cancel button in the middle, underneath the input box.

Chapter 4

Troubleshooting

4.1 Frequently Asked Questions

This is the section which should hopefully answer most of the questions that most users might have about the system. Sending emails to one of the addresses in the contact section in the beginning of the user manual may help you get your answer but it is best if you continue looking for an answer whilst you wait for an official reply.

4.1.1 What does Turtlenet do?

Think of Turtlenet in a similar manner to any other social network commonly in use. It allows users to communicate with each other and allowing other people to voice their opinions on what others have written. At the moment it is text based, meaning you can't attach images and video to it when you post or comment. You can however send links to such content to each other. That is a convenient enough work around for the time being as it means that no one is having to download an encrypted video but are never able to view it as they do not have the key to unlock the data. I think everyone will appreciate not having to download hundreds of copies of current top 40 each week.

4.1.2 How many accounts can I have on Turtlenet?

You should only need one, but we don't preclude you from having more. If you merely wish to seperate the content people can see then categories are a better solution, and future versions of Turtlenet will allowing the sharing of different personal information with different groups. If you simply must have multiple accounts though, there's nothing stopping you. Just launch the client in a different directory.

4.1.3 I forgot my password. Can someone reset it for me?

The short answer is no. Turtlenet was designed so that no one but the user had any access to their account. As a result, if you lose your password we are unable to recover anything in the account. The only thing you can do is simply to create another. Feel safe in the knowledge that everything is encrypted on your old account so at least no one can access what was lost except those people you already shared it with.

4.1.4 Where is everything stored?

Information is stored on your computer, laptop or whatever else it is that uses the Turtlenet client. Each client downloads all of the data and reads what it can, using keys you have collected over time off of other users. Keeping it local means that no readable is stored on the server, so evil moderators cannot have their way with your data. Encrypted data is stored on the server, but nobody can read it who you didn't send it to.

4.1.5 How big does this database get?

As the only things being stored are text, not images or video, this means that each message is only small and will likely be less than a few megabytes over one year's very active use.

4.1.6 Why would someone want to build from source?

Given that compatability of jars isn't an issue the only reason to do so is to ensure that your binaries derive from the public source code and not an evil secret version.

4.1.7 The Client does stuff I don't think it should do...

You may have found a bug for us accidentally. email to one of the addresses at the beginning of the user manual and the developers will have a look at it. As the source is being released, maybe the community will have a look and suggest a fix themselves.

4.1.8 What do Server Moderators of Turtlenet do?

We don't have any, we can't moderate content we can't see.

4.1.9 I want to mod Turtlenet. Can I have the source?

It's nice to know that others wish to take up the helm, pioneering a secure method of communication. You can have the source, it is available to the public to browse and modify.

4.1.10 Why choose 'X' over the clearly superior 'Y'?

As developers ourselves, we understand that other people have differing opinions. That's the joy of releasing code. Other people can pick up what we have done, or use our ideals as a starting point for their own thing. What this project stood for is ease of use for the end user and security from any unwanted external influences and this, we believe, is achieved.

Chapter 5

Side Notes

5.1 Java Database Connector

Our project uses the 'Java Database Connector' (JDBC) SQLite driver developed by 'Taro L. Saito' - also known as xerial. Xerial's website is here: http://xerial.org/
if you wish to obtain the driver or otherwise contact and give support, then there is your person.

Part IV

Portfolio

Chapter 33

Deviations in Requirements and Design

1. We have decided that the event function isn't very valuable and so dropped it from the requirements early in development.
2. We decided that having a website and active servers was important and so added it early in development.
3. Our data flow has changed in that the client now updates the local database without waiting for updates from the server to arrive. This was done so that network latency didn't interfere with the user experiance. All actions are still sent to oneself via the server, else multiple clients with the same key wouldn't function.
4. The client-client protocol has been significantly expanded so that all actions can be represented within it. This is so that if all one has is a keypair to an account, that account may be fully recovered. An example of new functionality in the protocol is that category creation and modification is recorded on the server (via encrypted messages sent, and viewable, soley to oneself). NB: The client-server protocol is wholly unaltered.
5. The datatypes in the database have changes due to the limitations of SQLite.
6. The primary key in many database tables has changed from an arbitrary value to a globally identifying cryptographic signature (from the message establishing the relevent datum.)
7. The database doesn't make use of foreign keys because the combination of network latency being potentially different for every message (due to Tor) and asymmetric relationships and communication means that foreign keys will often reference something that either doesn't exist

yet or will never exist. Furthermore in SQLite a foreign key can only reference one thing, and two of our potential foreign keys don't always reference the same field. Therefore there is only one possible foriegn key in our entire database: tCategoryMembers.catID references tCategory.catID, and even that is tenuous at best as relies upon undefined behaviour of the frontend. For these reasons we removed foreign keys from the schema. The new database design is as follow:

8. A number of accessors were added to the Message class for extracing information from different types of messages.

9. The DBStrings class was added so as to keep SQL query strings seperate from java code, and to localize them within one namespace. This class merely contains a large number of static strings.

10. The Logger, Tokenizer, MessageFactory, and F(ile)IO class were added as helper classes. Tokenizer was created instead of using javas existing Tokenizer class because we needed a tokenizer automatically convertable to javascript. A factory class was required because the Message class cannot contain constructors not automatically convertable to javascript.

11. The following data bearing classes were created to return structured data to the HTML/JS frontend. They allow the more powerful java backend to extract (and format) data from the database before returning a simple class containing it.

    (a) Friend

    (b) CommentDetails

    (c) Conversation

    (d) PostDetails

12. The Turtlenet, TurtlenetImpl, and TurtlenetAsync classes exist to provide an asynchronous interface between the HTML/JS frontend and the Java backend.

33.1 Logical table design version 2.0

tCategory

catId	canSeePDATA

tCategoryMembers

pk	catID	userKey

tClaim

sig	name	claimTime

tRevocations

key	sig	timeOfLeak	creationTime

tPostVisibleTo

pk	postSig	key

tConvoKeys

pk	convoID	key

tConvos

convoID	timeCreated

tUser

key	username	knowName	email	name	gender	birthday

tComment

sig	msgText	senderKey	parent	creationTime

tPost

sig	msgText	time	recieverKey	sendersKey

tConvoMessages

pk	convoID	sendersKey	msgText	time

tEvent

sig	startTime	endTime	creatorKey	accepted	name	creationTime

tLike

pk	likerKey	parent

33.2 Database design version 2.0

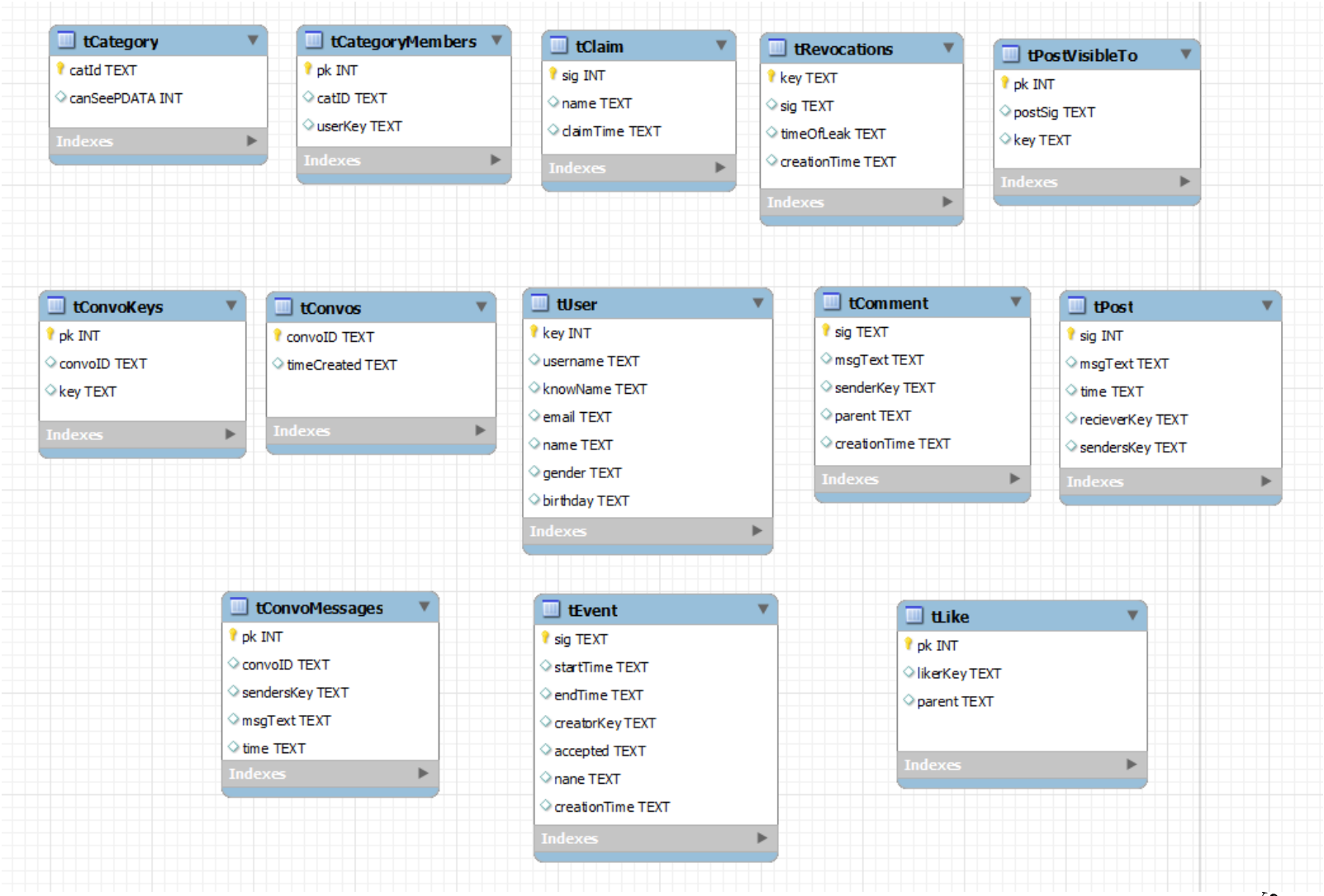


Figure 33.1: Database Entity Relationship diagram

33.3 Expanded Client-Client Protocol

Given number of new message types were added a summary of the new types can be found below:

Command	Format of content	Purpose
ADDCAT	true:family	add a new category named 'family' that can see your profile data
UPDATECAT	false:family	change the category named 'family' to be unable to see your profile data
ADDTOCAT	<key>:family	where '<key>' is replaced with a UTF-8, base64, X509 encoded RSA public key it adds that key to the category named family
REMFROMCAT	<key>:family	where '<key>' is replaced with a UTF-8, base64, X509 encoded RSA public key it removes that key from the category named family
ADDKEY	<key>	where '<key>' is replaced with a UTF-8, base64, X509 encoded RSA public key it adds that key to your list of known people

Chapter 34

Turtlenet Website

The website (www.turtlenet.co.uk), allows users to download the client and view details on the project. For example, who has been working on it, and the fine print regarding the licensing. These things can not only be useful, but also necessary to inform the user what the program is, and why they need it, along with providing a suitable means of distribution for the project. Below is a short description of the website and its content.

- Home Page: This is a quick general description about what Turtlenet is, providing you with a link to both get the sourcecode for Turtlenet, and the client itself.
- Downloads: Holds the Turtlenet.jar file link, allowing the user to retrieve the client, as it does other pertinent information regarding any dependencies the client will need to run.
- About Us: This page is a small area dedicated to the team behind Turtlenet, Ballmer Peak, and our views on the project.
- Content: This area of the website contains all the contact information for our staff, should you need to contact them.
- FAQs: Frequently Asked Questions about the project come here.
- License: Finally, here is the legal rambling regarding the license for the project.

Chapter 35

Hosting

We chose Amazon Web Services(AWS) to host the remote server component of Turtlenet and the project website. The primary reason for this is that AWS offers a free package for the first 12 months of membership. The price after this is also fairly competitive although we would probably consider other providers if the offer didn't exist.

Another advantage of AWS is, due to their size, Amazon have many servers available which lends itself to high performance and increased redundancy. Performance isn't a major issue as the Turtlenet remote server acts as little more than a middleman. Still, the lower the latency the better.

It is uncertain if Amazon has provided any of it's customers personal data to the NSA or any other government sponsored agency. This isn't of concern though as very little data is stored by the Turtlenet remote server. Due to the distributed nature of Turtlenet all sensitive information is stored in a local database on the users device.

Chapter 36

Source Code and Github.com

We chose to store our source code in a Git repository hosted by Github at: `https://github.com/thomal/COMP208/`.

The repository records all changes to files which allows group members know who has modified files, what has been added and what has been deleted.

If more than one person edits the same file it is possible to merge together the work of each person, often automatically. When more than one person edits the same line manual intervention is required to see who's edit should be kept.

We can also look back at previous revisions of files to figure out where bugs were introduced or to resurrect old code that has since been removed but may still be useful.

Chapter 37

Testing

37.1 Automated Testing

We have included a series of automated tests with Turtlenet. These tests are ran automatically whenever the binaries are built, they can be ran manually with 'make test'.

For example we can test the part of the system responsible for parsing the contents of wall posts by constructing a wall post Message instance and calling POSTgetKeys(). If we do not receive the expected value for any test the developer is alerted.

More examples of the automated testing we used can be found at: `https://github.com/thomal/COMP208/blob/master/src/ballmerpeak/turtlenet/testing/Test.java`.

37.2 Black-box Testing

Black-box testing comprised mainly of:

- Clicking on buttons even if they didn't fit into the logical work flow.
- Attempting to 'cheat' the system by making it it do things that aren't intended by the design.
- Entering silly values into various parts of the system in an attempt to see if this would break anything.

The system functioned as expected in all cases. Our intent in not validating the date field was because people don't read things telling them how to format dates but rather type them how they normally do; however it allows for peoples date of birth to be "the past". This perhaps isn't such a bad thing, but was ultimitly unintentional.

37.3 Usability Testing

We had a person who had never used Turtlenet before use the system with only five minutes of introduction beforehand. Without further instruction she was able to easily register, add my public key that I sent her, and take part in a conversaiton. She did however need help finding the wall of a friend. After I showed her she found it easy to do so after.

From this we've concluded that it's preferable to have more obvious ways of accessing anothers wall. This test occurred too late in development to add the change.

Chapter 38

Future Development

38.1 Interface Framework

Currently the interface framework is the Google Web Toolkit (GWT). This currently allows the Turtlenet client's interface to run in a web browser of the user's choosing. This choice helped the group and the project initially by allowing the developers to not consciously worry about any particular problems in the running of the project in terms of the front end - our mentality was that everyone had a browser on their computers so it was one less requirement to add to the minimum requirements section of the user manual. In hindsight however, what this decision has done is open the project to a flood of potential bug reports; some from inexperienced users hoping for a fix on Internet Explorer whilst others filling in reports about how their personally compiled browser is not rendering a button, for example. With Java, the code is run by a Java Virtual Machine (JVM) so there is only one platform to worry about whereas through the use of a browser being a container for the user interface, each with different layout engines and capabilities, the project can expect to receive bug reports from at least four different front ends.

With the potential workload heavily increased just by a decision made by the user, which we assume holds little to no knowledge and just wishes the project work on their computer, this may be a problem for the future developers of Turtlenet. As a possible solution, a different framework could be used which provides an executable that is tailored to the operating system, as opposed to a web browser. Running in a native window as opposed to a browser means that there is only a couple of different ways the front end can appear, variations mainly appearing due to a change of operating system - although a different desktop environment (DE) would also affect the final look of Turtlenet. This improves the usability of the project as the final outcome would be an executable file, which runs in the same or similar manner to other programs on their computer, improves consistency as GWT creates JavaScript as a core component in the running of the front-end, which may be blocked

by the browser as well as creating specific statements which are interpreted or ignored dependent on the browser currently in use. Finally there is debugging, which is improved because there are fewer main variables for the testers to cater to, such as versions of browsers and their plug-in files which may hamper use of the front end. Through consolidation into a native window, testing and bug catching can become more efficient as there are less programs to load, and less time required.

38.2 Interface and House Style

The current interface for the client has been made in such a way that it provides all of the functionality of the project in fairly easily identifiable sections. What it does not do though is look polished enough to be on an average user's computer yet. This is most likely due to time constraints stressing for functionality as opposed to aesthetics. Another potential problem is the house style of the front-end. Green has symbolic meaning and was not chosen simply due to the name of the project - Turtles more often than not have darker colours such as brown or grey and not green. Green is often used in healthcare as a sign that something is either safe or good for you (the green health 'plus' being an example), which is what Turtlenet aims to be for your communicative efforts. On a per-user basis however, colour is simply a way of making the front end become more pleasant.

This leads to the problem of personal taste - some people don't like green. Therefore to increase usability of the project as well as the total amount of users, themes could be a future development. Allowing the user to change the look of the front end can make a difference to the amount of people using the project. More users may appear if the project synchronises well with the rest of their system.

38.3 Languages Used

The project used Java for the back end of the system, SQLite for the Database and Java converted to JavaScript for the front end. Java was chosen for the interoperability of the language - being able to run on whatever has a Java Virtual Machine (JVM), which are available for most operating systems. Most users have the Java Runtime Environment (JRE) installed, which includes JVM so Java was a good choice for the project.

SQLite is a notably lightweight Database Management System (DBMS) at the expense of some features that are used in a more complete SQL solution, none of which were needed for the project. SQL notably requires you to define data type as well as the length of the variable as well - while SQLite is more lenient in this regard, removing this constraint completely would make a system more usable. An example of a more user friendly database would be one that uses MongoDB but that is not as popular as an SQL derived DBMS, so this is the reason SQLite was chosen.

Google Web Toolkit (GWT) allowed one of our developers the capability of writing code in a similar manner to Java which when compiled creates the required JavaScript and Ajax code. On a technical level we believed this to be quite clever, and was one of the reasons we chose GWT for the interface framework. In hindsight it would have been better to choose a different framework which would allow us to get a native executable after compilation. This would mean that the user does not need to open a terminal and enter any Java commands, improving usability by not forcing the user to enter an environment that they are not comfortable with.

Chapter 39

Ballmer Peak

Ballmer Peak is the team behind Turtlenet, both its designers and creators. We have worked together over the past 6 months to create a new, innovative product the entire team is proud of. Pooling our resources and individual technical skills we feel we came together as a team and produced a quality product. We all had our roles to fill in this project:

39.1 Peter Duff - Team Lead

- In charge of project management and organisation, as well as assigning roles and responsibilities, amongst other generic managerial admin.
- Was the designer and creator of the Turtlenet website.
- Proofreader of the entire requirements, design and portfolio documents.

39.2 Luke Thomas - Lead Programmer

- Designed and created the initial prototype and idea for the project.
- Wrote the whole of the client, except for Database.java, DBStrings.java, and frontend.java. Significantly contributed to Database.java and frontend.java.
- Wrote the whole of the server.
- Wrote the automated tests, designed LaTeX documents.

39.3 Louis Prince - Programmer

- Programmed and designed the GUI for the client (frontend.java).
- Redesigned parts of the Turtlenet website.

39.4 Aishah Senin - Database Developer

- Documented all aspects of the database.
- Designed and created the SQLite database from scratch.
- Wrote database queries.

39.5 Michael Chadwick - Software Engineer

- Wrote the majority of the user manual for the program.
- Provided heavy documentation regarding some of the finer, non-technical points of the project (Gantt, Risk Asses. etc.).
- Wrote a significant portion of Database.java

39.6 Leon Choi - Researcher

- Main researcher into existing similar systems and potential future work.
- Designed a mock GUI for future android work.

Chapter 40

Achievement of Objectives

We successfully produced a usable social network requiring minimal knowledge of cryptography that can be used to hide both what one says and whom one says it to. We did not however allow users to transfer public keys as QR codes, rather we made them copy/paste a string of text. As an extra feature we added Tor support to hide the originating IP addresses of traffic, while this isn't as secure as our other measures it is secure in practice, with the NSA being unable to de-anonymise Tor users[**torStinks**]. Our pages do not auto-refresh at the moment, meaning that to see new content a user must reload the page manually. On the other hand we added the feature of encrypting all information stored locally, and extended the protocol so that with only a users keypair their entire local database may be re-derived on any device; this is useful for users with multiple devices.

On the whole the missed some objectives and added a good deal more. We feel that what we gained was far greater than what we lost, and so was worth the time we spent on it in lieu of the unimplemented, but planned, features.

Chapter 41

Evaluation

I think the best evaluation is that we didn't expect this. This was a thoroughly challenging project, that we did not anticipate being so large when we started. Typically as a student, we found university modules to have a similar workload to each other through the university courses so far. This group project however was a different thing entirely. The work hours put in for this module eclipsed all other modules by far, even combined in some parts. This being said, it has also been rewarding to many of us, to work together producing a much bigger result than the small projects we normally create. Self learning was also widely needed during this module in order to teach us all the skills required to complete the project.

Overall though, we feel the project has been a great success. With a live website running right now (www.turtlenet.co.uk), live working repository (https://github.com/thomal/COMP208), a product that works fully functioning as intended, its hard not for us to be satisfied. Even despite not meeting all our original requirements, we produced a functional social network that aptly serves to complete our original target goal of a secure social network, that holds privacy in its highest regard. To say things went perfectly however, would be a fallacy.

There are areas we could have definitely improved upon, such as the general teamwork between members. The idea of group study sessions unfortunately came too late in the project to be done, but the idea of us all taking a weekend to work on the project together on one desk would have likely yielded positive results. Better work deadlines would have also been useful, more frequent, smaller workloads may have well served better than larger apart ones. We should have also done more research on what to produce the client UI with, as it turns out GWT was not the best choice. Apparently it is quite limiting to use and better alternatives would have been a more optimal choice.

We did not make all bad decisions though, the idea of abandoning the provided university workspace in favour of git was an astounding success, and we are all convinced reproducing the same project in that workspace would simply not be possible due to the complexity of the program.

www.github.com provided us with the means to make the project possible, and documents our project from the start step by step, an excellent tool for group projects such as this. We are all in agreement version control is a must for group projects of this scale.

In summary, the project was successful, and something for us as a team to be proud of. It is something many of us wish to continue working on even after this module is all over. It is something that taught us more skills than any other module. And most of all it is something for all to have, a gift to the world pool of knowledge, accessible to all at no cost, and perhaps one step forward in a fair, free internet.

Appendices