COMP208 - Group Software Project
Ballmer Peak

M. Chadwick; Choi, S.F; P. Duff; L. Prince; A.Senin; L. Thomas

March 14, 2014

Contents

Appendices

Part I

Design

Chapter 1

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 2

Architecture

2.1 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

2.2 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.

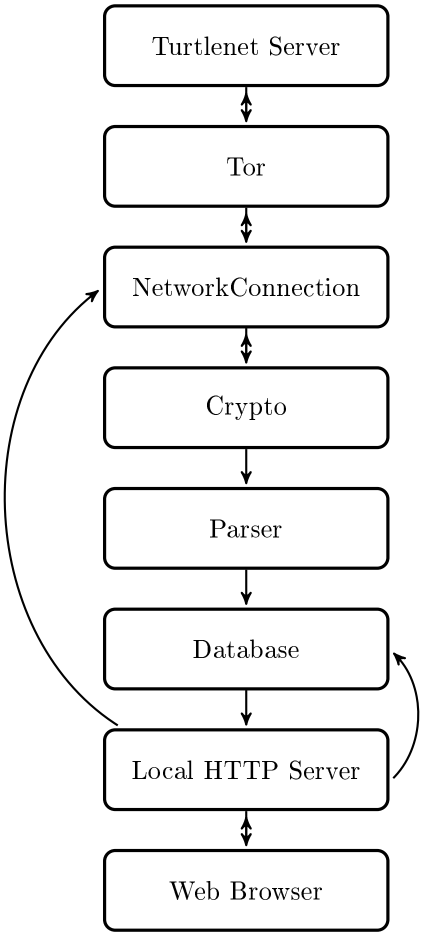


Figure 2.1: Module Interaction

Chapter 3

Data Flow Diagram

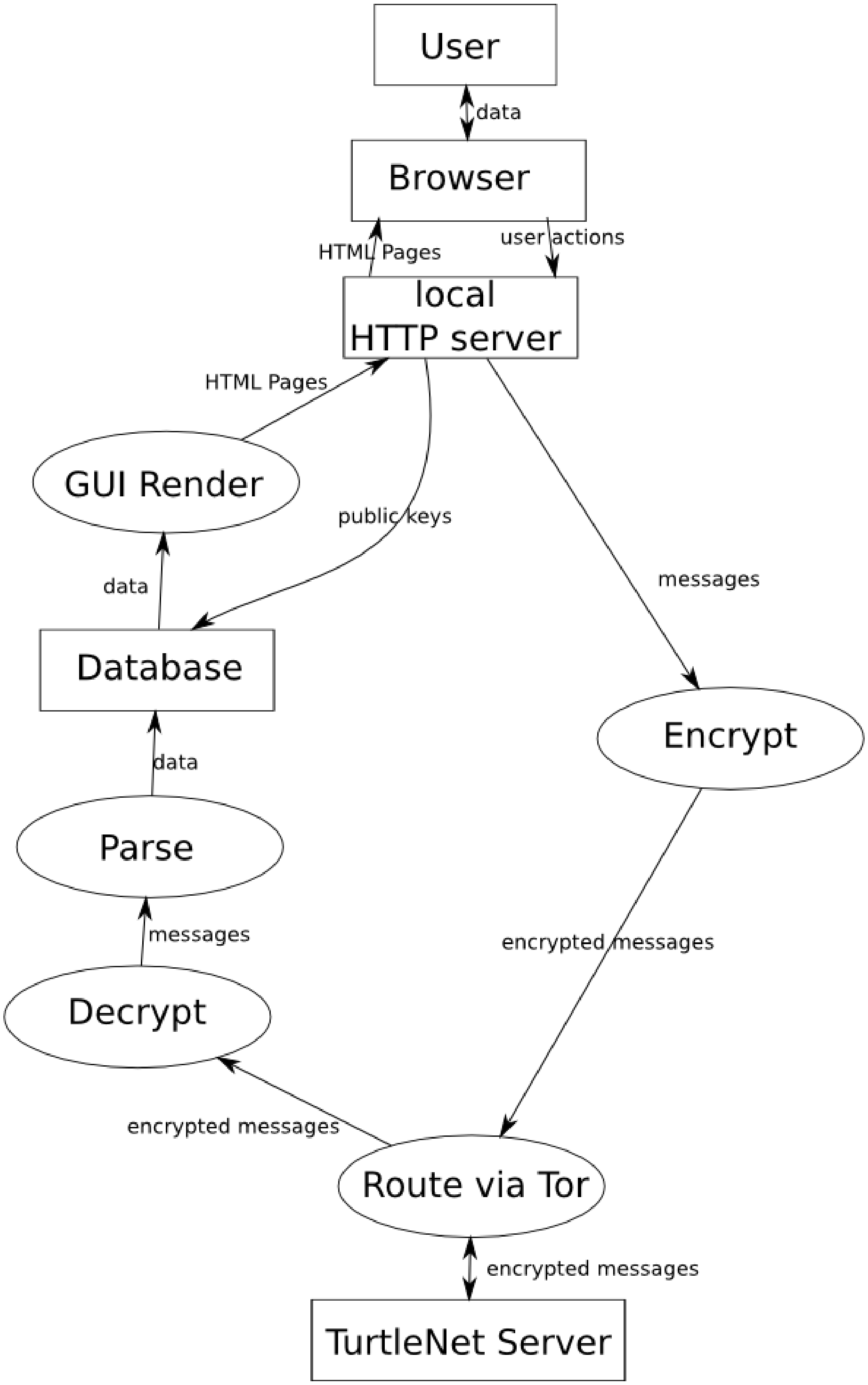
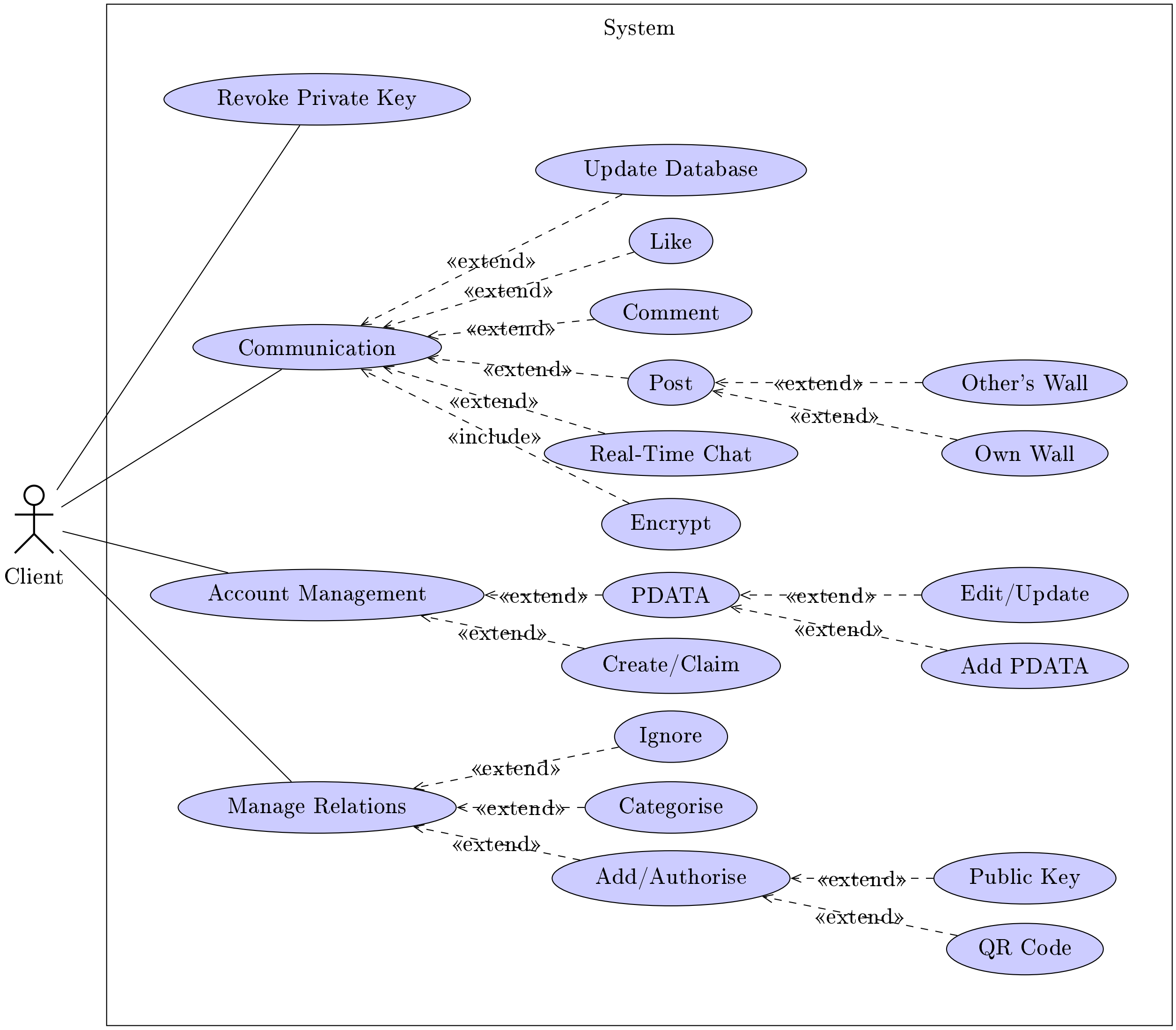


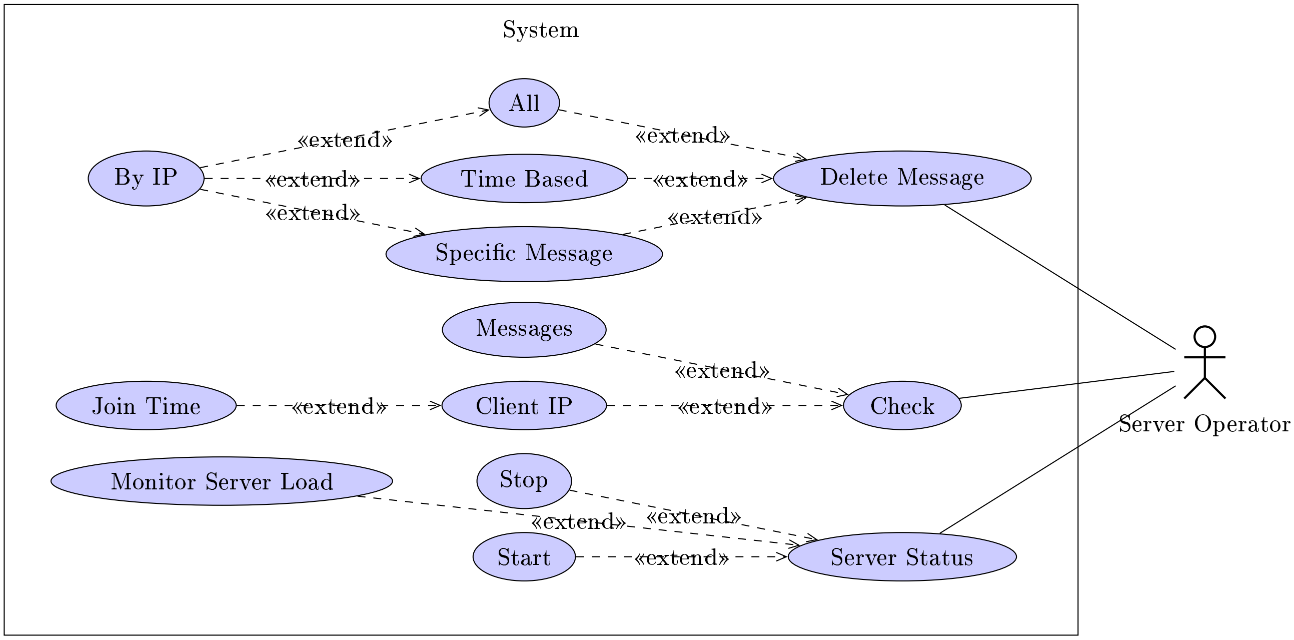
Figure 3.1: Data Flow Diagram

Chapter 4

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


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

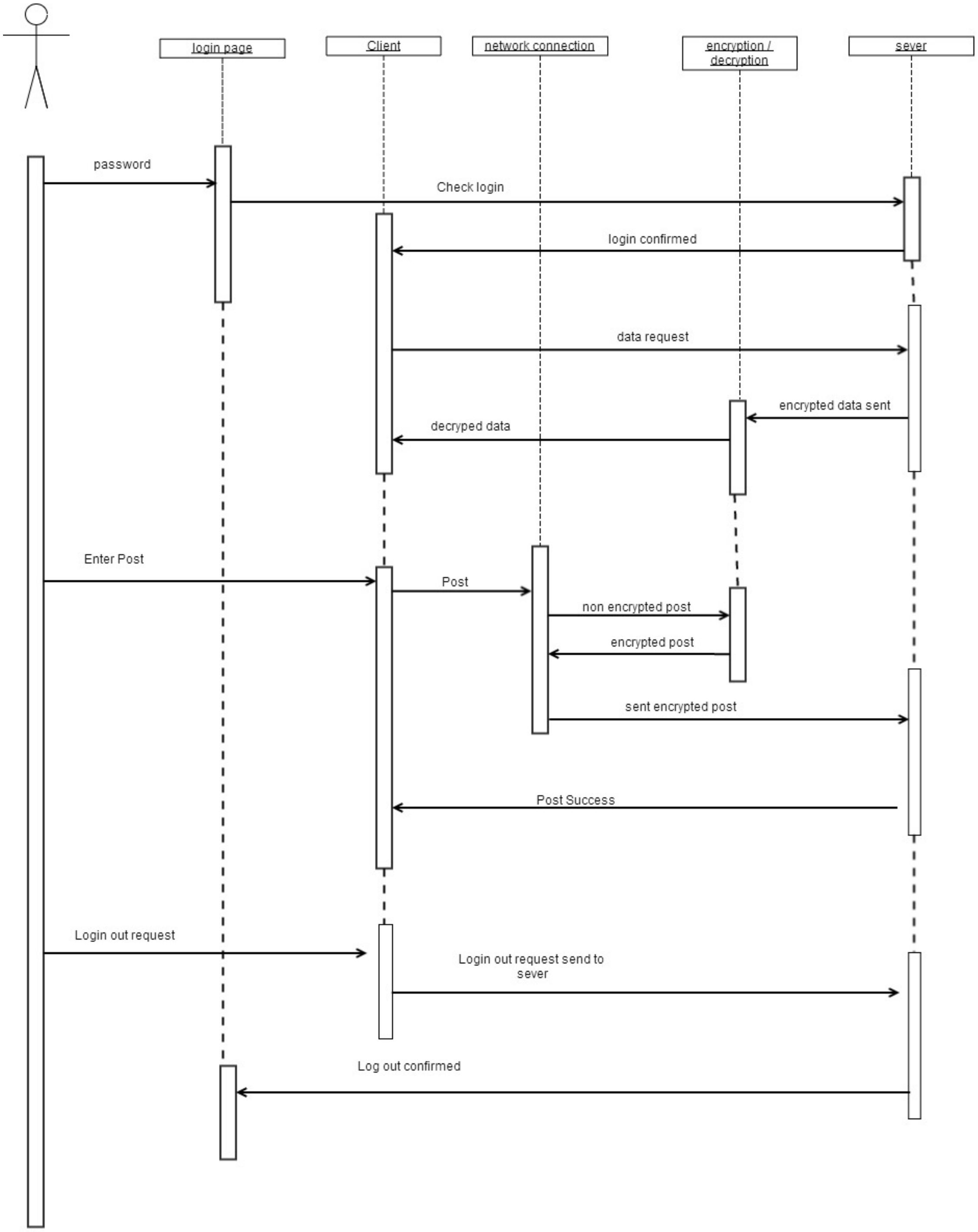


Figure 4.1: Sequence Diagram

Chapter 5

Protocol

5.1 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.

5.2 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 5.1: 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 $<unix\_time\_in\_ms>\_<SHA256\_hash>$.

5.3 Client-Client Protocol

5.4 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.

5.5 Message Formatting

5.5.1 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:

$$<command>\backslash<signature>\backslash<content>\backslash<timestamp>$$

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

An example follows:

$$\text{POST}\backslash<signature>\backslash\text{Hello, World!}\backslash 1393407435547$$

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.

5.5.2 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.

5.6 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:

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

5.7 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:

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

5.8 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*>; ...

An example follows:

PDATA\<*signature*>\name:Luke Thomas;dob:1994;\<*timestamp*>

5.9 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:

CHAT\<*signature*>\<*keys*>\<*timestamp*>

The format for keys follows:

<*key*>: <*another_key*>...

An example follows:

CHAT\<*signature*>\<*key1*>:<*key2*>\<*timestamp*>

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

PCHAT\<*signature*>\<*conversation*>:<*message*>\<*timestamp*>

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

PCHAT\<*signature*>\9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08:First!\<*timestamp*>

5.10 Posting to own wall

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

POST\<*signature*>\<*message*>\<*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":

POST\<*signature*>\Hello, World!
foo\\bar\\baz\<*timestamp*>

5.11 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).

5.12 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*>

5.13 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*>

5.14 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:

$\text{EVNT}\backslash<signature>\backslash<event\_start\_time>\text{: }<event\_end\_time>\text{: }<event\_name>\backslash<timestamp>$

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:

$\text{EVNT}\backslash<signature>\backslash\text{1389657600000: 1389744000000: bobs birthday}\backslash\text{1388676821000}$

Chapter 6

Class Interfaces

6.1 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 6.1: Server

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

Table 6.2: 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 6.3: 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 6.4: Crypto

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

Table 6.5: 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 6.6: 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 6.7: 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 6.8: 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 6.9: Pair<A, B>

6.2 Class Diagram

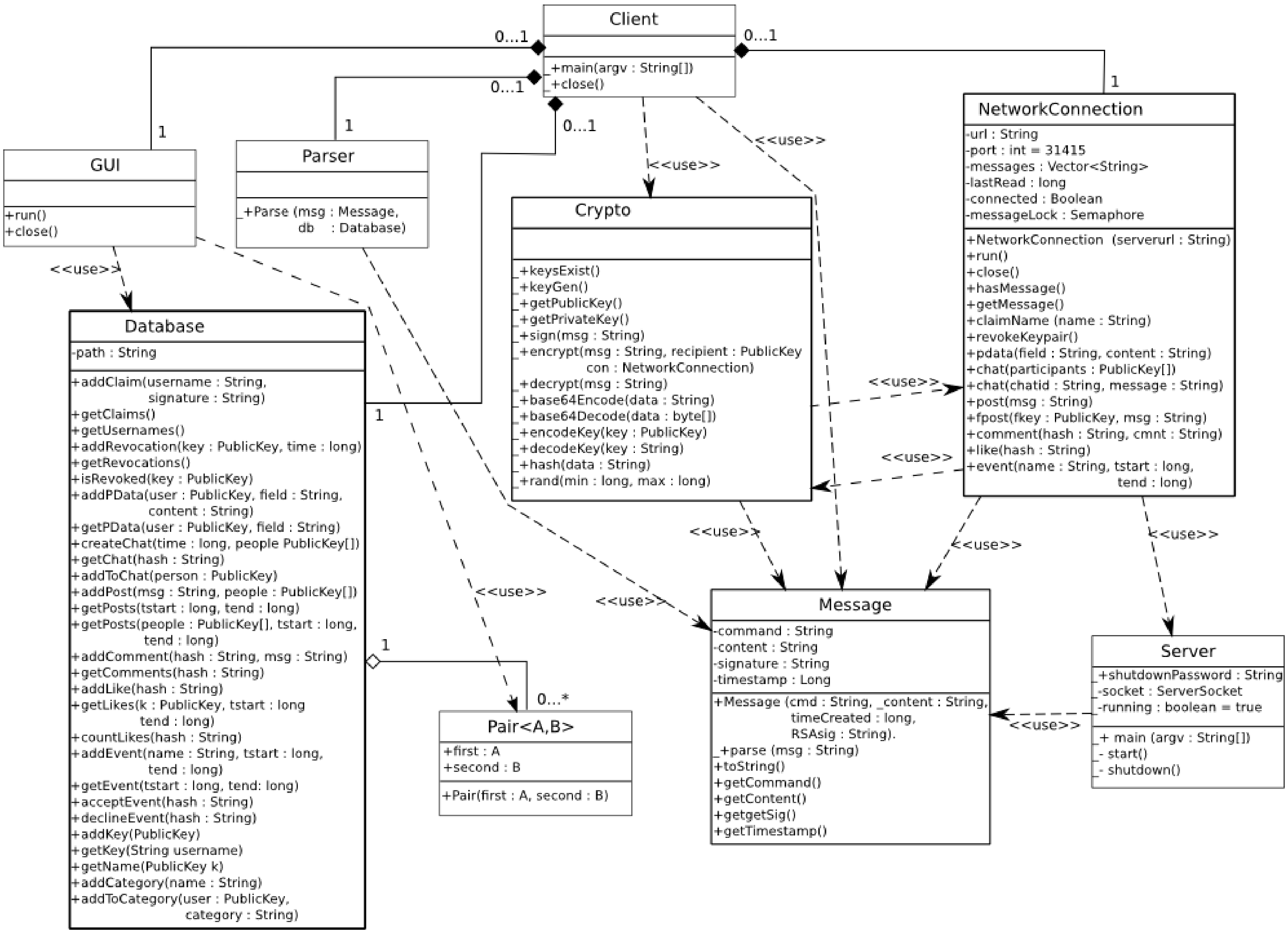

Client
+main(argv : String[])
+close()
0...1
0...1
0...1
0...1
1
1
1
1
GUI
+run()
+close()
Parser
+Parse (msg : Message,
db : Database)
NetworkConnection
-url : String
-port : int = 31415
-messages : Vector<String>
-lastRead : long
-connected : Boolean
-messageLock : Semaphore
+NetworkConnection (serverurl : String)
+run()
+close()
+hasMessage()
+getMessage()
+claimName (name : String)
+revokeKeypair()
+pdata(field : String, content : String)
+chat(participants : PublicKey[])
+chat(chatid : String, message : String)
+post(msg : String)
+fpost(fkey : PublicKey, msg : String)
+comment(hash : String, cmnt : String)
+like(hash : String)
+event(name : String, tstart : long,
tend : long)
Crypto
+keysExist()
+keyGen()
+getPublicKey()
+getPrivateKey()
+sign(msg : String)
+encrypt(msg : String, recipient : PublicKey
con : NetworkConnection)
+decrypt(msg : String)
+base64Encode(data : String)
+base64Decode(data : byte[])
+encodeKey(key : PublicKey)
+decodeKey(key : String)
+hash(data : String)
+rand(min : long, max : long)
Database
-path : String
+addClaim(username : String,
signature : String)
+getClaims()
+getUsernames()
+addRevocation(key : PublicKey, time : long)
+getRevocations()
+isRevoked(key : PublicKey)
+addPData(user : PublicKey, field : String,
content : String)
+getPData(user : PublicKey, field : String)
+createChat(time : long, people PublicKey[])
+getChat(hash : String)
+addToChat(person : PublicKey)
+addPost(msg : String, people : PublicKey[])
+getPosts(tstart : long, tend : long)
+getPosts(people : PublicKey[], tstart : long,
tend : long)
+addComment(hash : String, msg : String)
+getComments(hash : String)
+addLike(hash : String)
+getLikes(k : PublicKey, tstart : long
tend : long)
+countLikes(hash : String)
+addEvent(name : String, tstart : long,
tend : long)
+getEvent(tstart : long, tend: long)
+acceptEvent(hash : String)
+declineEvent(hash : String)
+addKey(PublicKey)
+getKey(String username)
+getName(PublicKey k)
+addCategory(name : String)
+addToCategory(user : PublicKey,
category : String)
1
1
0...*
Pair<A,B>
+first : A
+second : B
+Pair(first : A, second : B)
Message
-command : String
-content : String
-signature : String
-timestamp : Long
+Message (cmd : String, _content : String,
timeCreated : long,
RSAsig : String).
+parse (msg : String)
+toString()
+getCommand()
+getContent()
+getgetSig()
+getTimestamp()
Server
+shutdownPassword : String
-socket : ServerSocket
-running : boolean = true
+ main (argv : String[])
- start()
- shutdown()
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>

Figure 6.1: UML Class Diagram

Chapter 7

Pseudocode

7.1 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()
}
```

7.2 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)
}
```

7.3 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)
}
```

7.4 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)
}
```

7.5 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);
}
```

7.6 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 8

Database

8.1 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.

8.1.1 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.

8.1.2 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.

8.1.3 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.

8.1.4 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.

8.1.5 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.

8.1.6 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.

8.1.7 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.

8.1.8 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.

8.1.9 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.

8.1.10 message_claim table

When the main user has not required any public key from the other user he is intended to add, the details will be entered onto this table until which a pubic key is retrieved. When retrieved, this information found in this table will be deleted and transferred into the *user* table along with other information that comes with it.

8.1.11 key_revoke table

In times when malicious activity might have been conducted, the users of Turtlenet is able to revoke their relations signature key. A signature suggests that this post is written by this person in particular, so when it is revoked, whatever that has been posted with this signature is considered false. This will inform the user not to trust any information that has been posted with this signature in particular.

8.1.12 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.

8.2 Table layout of the database

NB: Public keys are 217 characters long, all id's are auto-incremented.

Table 8.1: user

Name	Datatype	Key
user_id	INT	PK
username	VARCHAR(25)	
name	VARCHAR(30)	
birthday	DATE	
sex	VARCHAR(1)	
email	VARCHAR(30)	
public_key	VARCHAR(600)	PK

Table 8.2: is_in_category

Name	Datatype	Key
is_in_id	INT	PK
category_id	INT	FK
user_id	INT	FK

Table 8.3: category

Name	Datatype	Key
category_id	INT	PK
name	VARCHAR(30)	

Table 8.4: private_message

Name	Datatype	Key
message_id	INT	PK
from	INT	
content	VARCHAR(50)	
time	DATE	

Table 8.5: is_in_message

Name	Datatype	Key
is_in_id	INT	PK
time	DATETIME	
message_id	INT	FK
user_id	INT	FK

Table 8.6: wall_post

Name	Datatype	Key
wall_id	INT	PK
from	INT	FK
to	INT	FK
content	VARCHAR(50)	
time	DATETIME	

Table 8.7: allowed_to

Name	Datatype	Key
allowed_to_id	INT	PK
user_id	INT	FK
category_id	INT	FK
post_id	INT	FK

Table 8.8: has_comment

Name	Datatype	Key
comment_id	INT(100)	PK
comment_content	VARCHAR(50)	
post_id	INT(100)	FK
user_id	VARCHAR(50)	FK
comment_comment_id	INT(100)	FK
time	DATETIME	

Table 8.9: has_like

Name	Datatype	Key
like_id	INT	PK
post_id	INT	FK
user_id	INT	FK
comment_id	INT	FK
time	DATETIME	

Table 8.10: events

Name	Datatype	Key
event_id	INT	PK
title	VARCHAR(10)	
content	VARCHAR(40)	
time	DATETIME	
start_date	DATETIME	
end_date	DATETIME	
from	INT	FK

Table 8.11: is_invited

Name	Datatype	Key
is_invited_id	INT	PK
user_id	INT	FK
is_in_category_id	INT	FK
event_id	INT	FK
decision	BIT	

Table 8.12: login_logout_log

Name	Datatype	Key
log_id	INT	PK
login_time	DATETIME	
logout_time	DATETIME	

Table 8.13: key_revoke

Name	Datatype	Key
revoke_id	INT	PK
signature	VARCHAR(45)	
time	DATETIME	

Table 8.14: message_claim

Name	Datatype	Key
username	VARCHAR(25)	PK
signature	VARCHAR(45)	

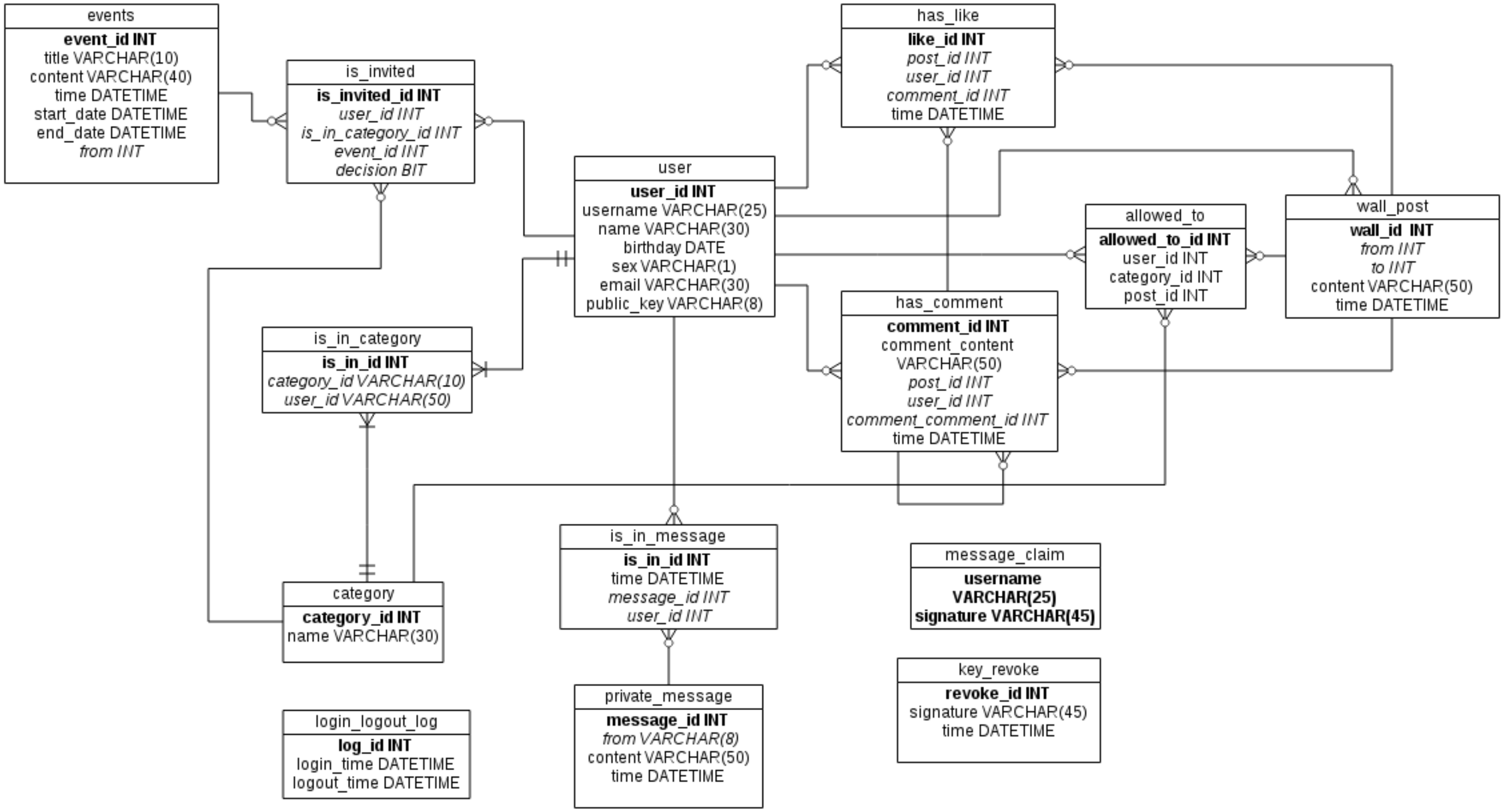


Figure 8.1: Database Entity Relationship diagram

Chapter 9

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 10

User Interfaces

10.1 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.

10.1.1 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.

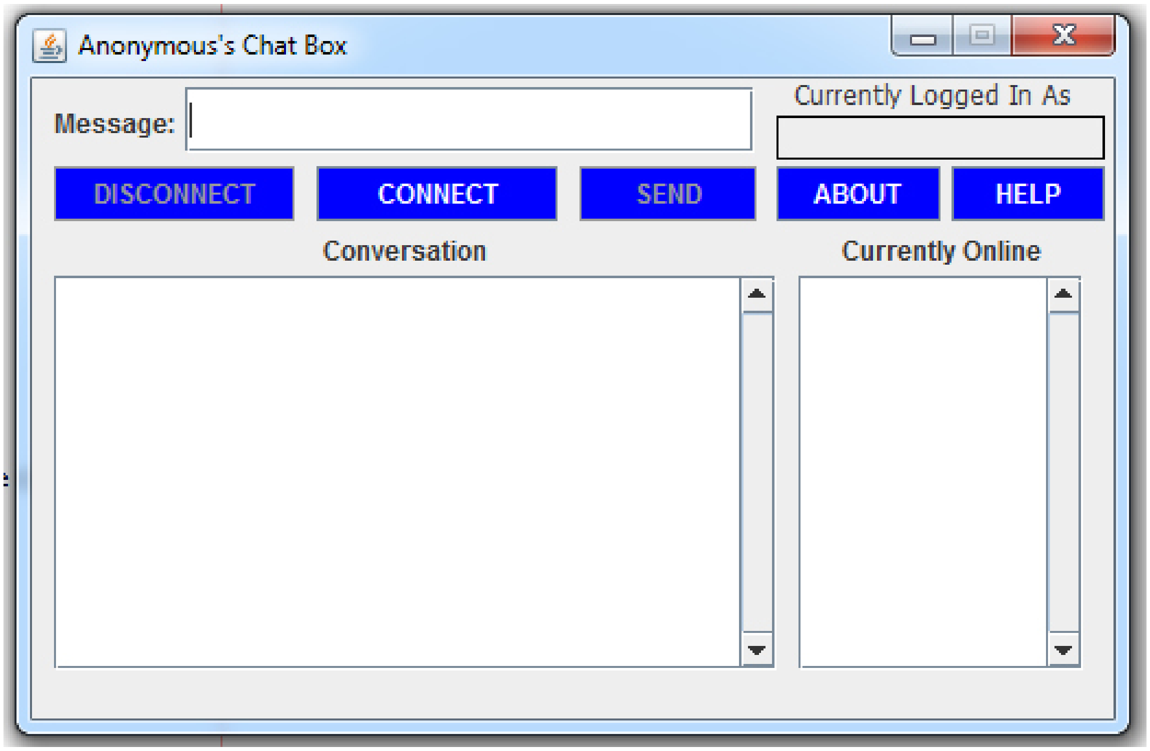


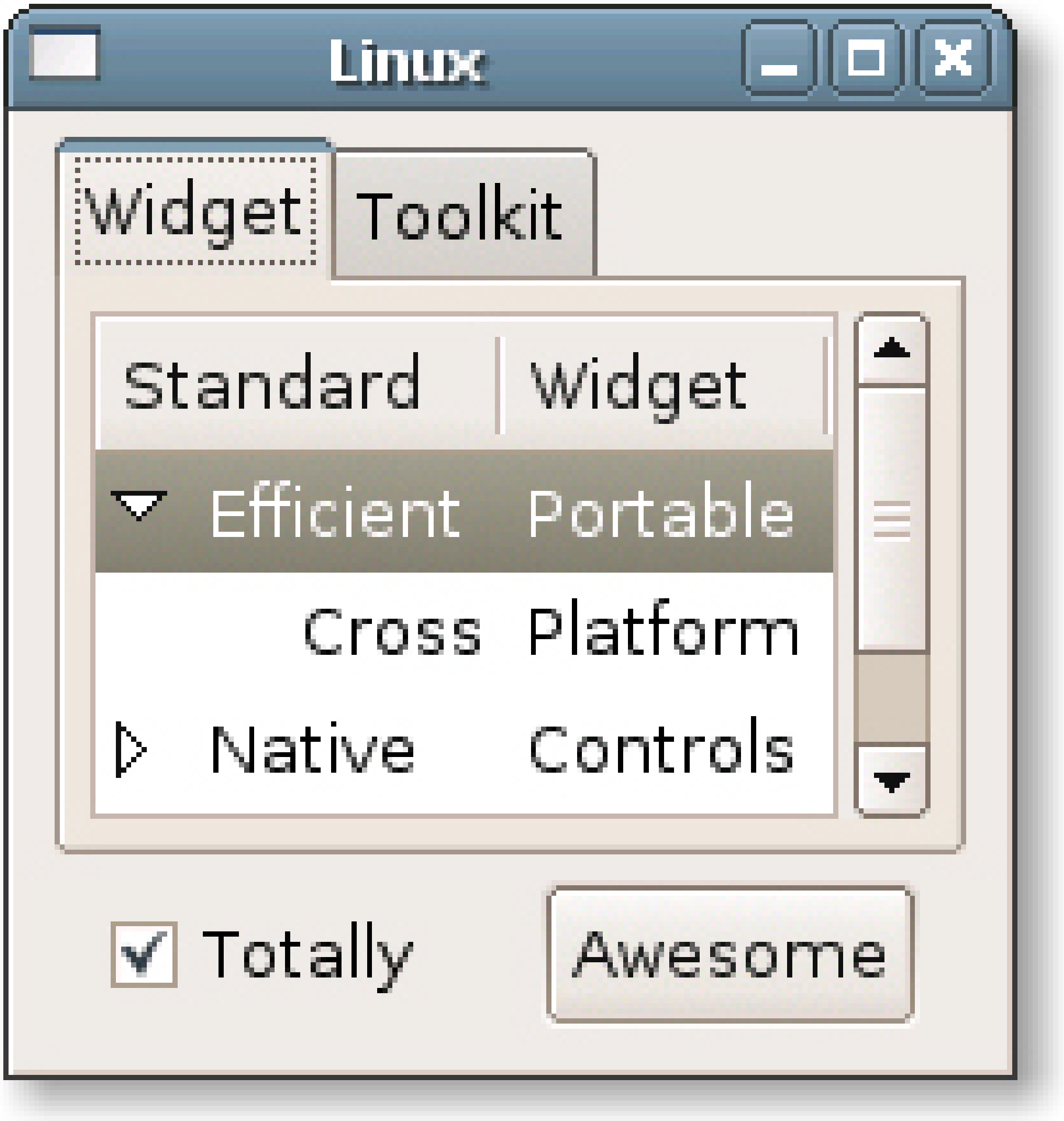
Figure 10.1: Swing Instant Messaging Application

10.1.2 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.

10.1.3 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.



Standard	Widget
▽ Efficient	Portable
Cross	Platform
▷ Native	Controls

Figure 10.2: 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.

10.1.4 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.

10.1.5 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.

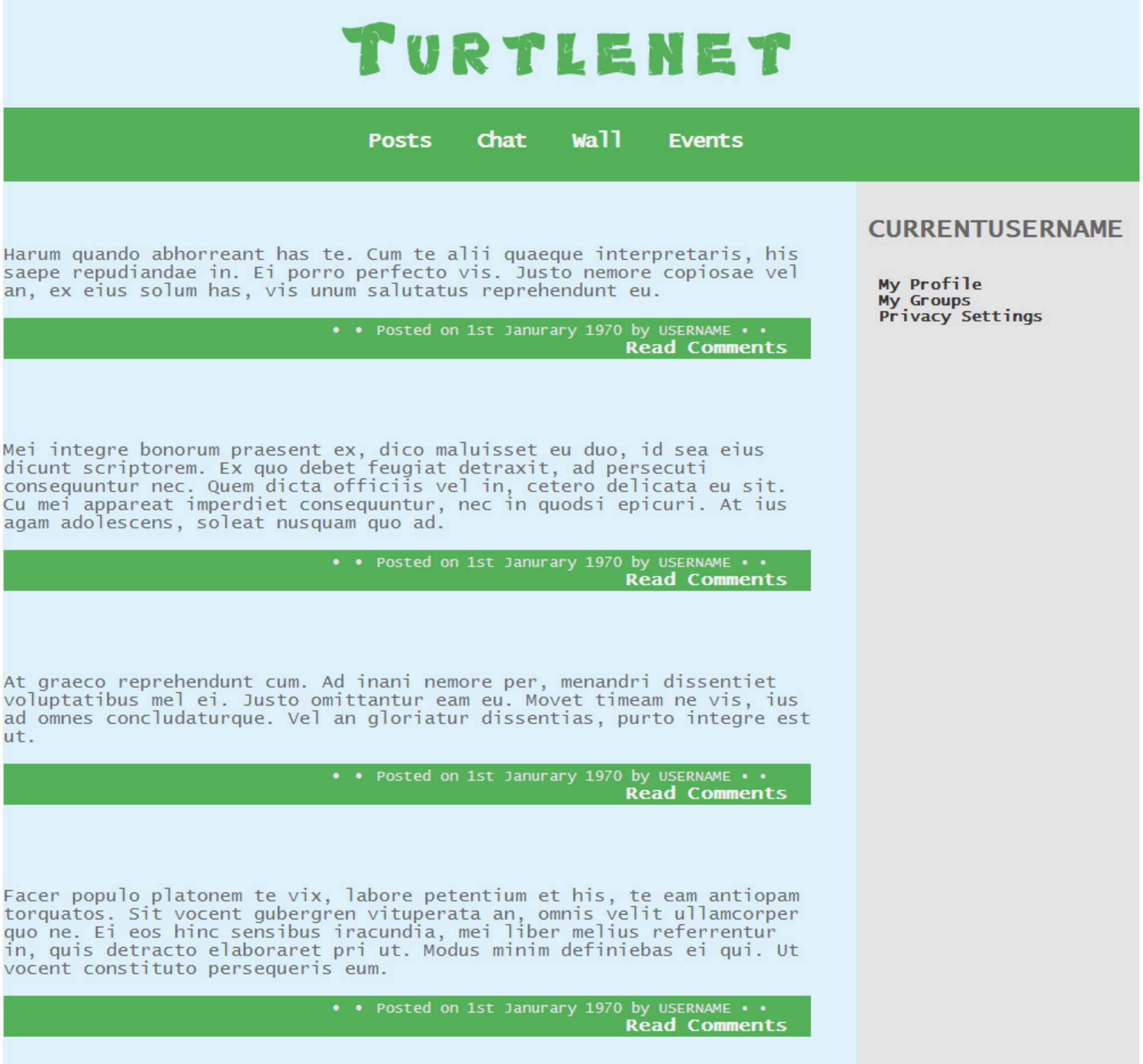
One disadvantage of this approach is the difficulty in implementing the required security measures and encryption in Javascript. This can be remedied by using a Javascript library such as the Forge project which implements many cryptography methods.

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.

10.2 GUI Design

10.2.1 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.



Harum quando abhorreant has te. Cum te alii quaeque interpretaris, his saepe repudiandae in. Ei porro perfecto vis. Justo nemore copiosae vel an, ex eius solum has, vis unum salutatus reprehendunt eu.

• • Posted on 1st Janurary 1970 by USERNAME • •
Read Comments

Mei integre bonorum praesent ex, dico maluisset eu duo, id sea eius dicunt scriptorem. Ex quo debet feugiat detraxit, ad persecuti consequuntur nec. Quem dicta officiis vel in, cetero delicata eu sit. Cu mei appareat imperdiet consequuntur, nec in quodsi epicuri. At ius agam adolescens, soleat nusquam quo ad.

• • Posted on 1st Janurary 1970 by USERNAME • •
Read Comments

At graeco reprehendunt cum. Ad inani nemore per, menandri dissentiet voluptatibus mel ei. Justo omittantur eam eu. Movet timeam ne vis, ius ad omnes concludaturque. Vel an gloriatur dissentias, purto integre est ut.

• • Posted on 1st Janurary 1970 by USERNAME • •
Read Comments

Facer populo platonem te vix, labore petentium et his, te eam antiopam torquatos. Sit vocent gubergren vituperata an, omnis velit ullamcorper quo ne. Ei eos hinc sensibus iracundia, mei liber melius referrentur in, quis detracto elaboraret pri ut. Modus minim definiebas ei qui. Ut vocent constituto persequeris eum.

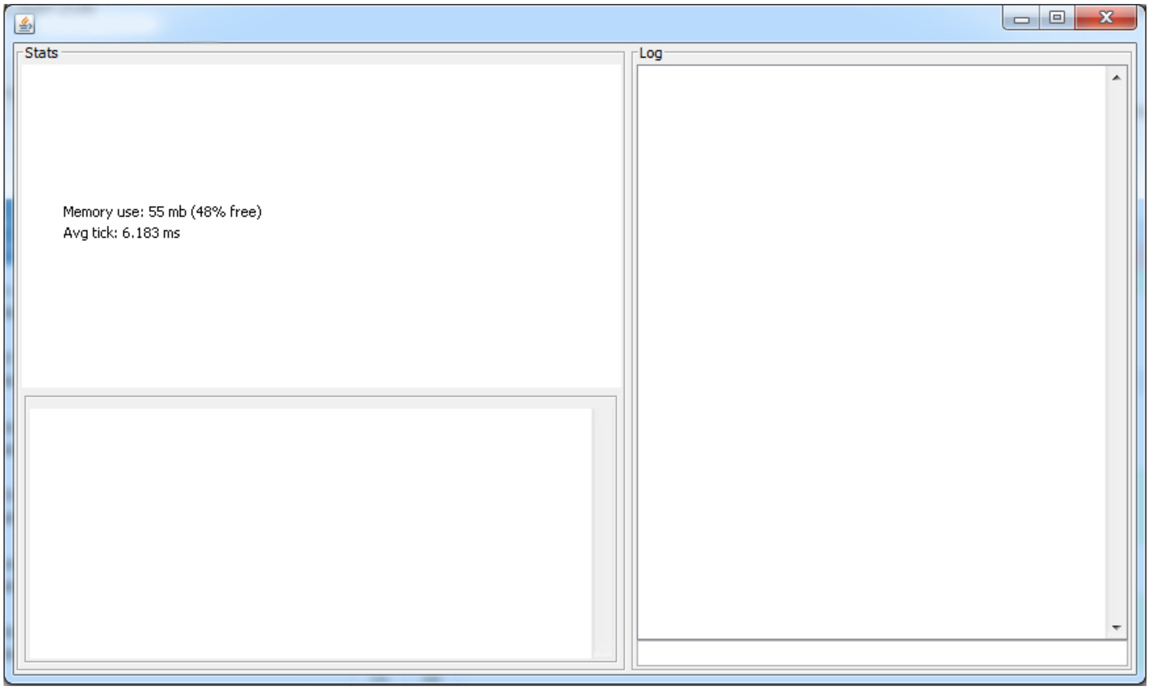
• • Posted on 1st Janurary 1970 by USERNAME • •
Read Comments

Figure 10.3: 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.

10.2.2 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.



Memory use: 55 mb (48% free)
Avg tick: 6.183 ms

Figure 10.4: Server Design Image

It should further be noted, 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 if the operator is a user itself.

10.3 Future Work

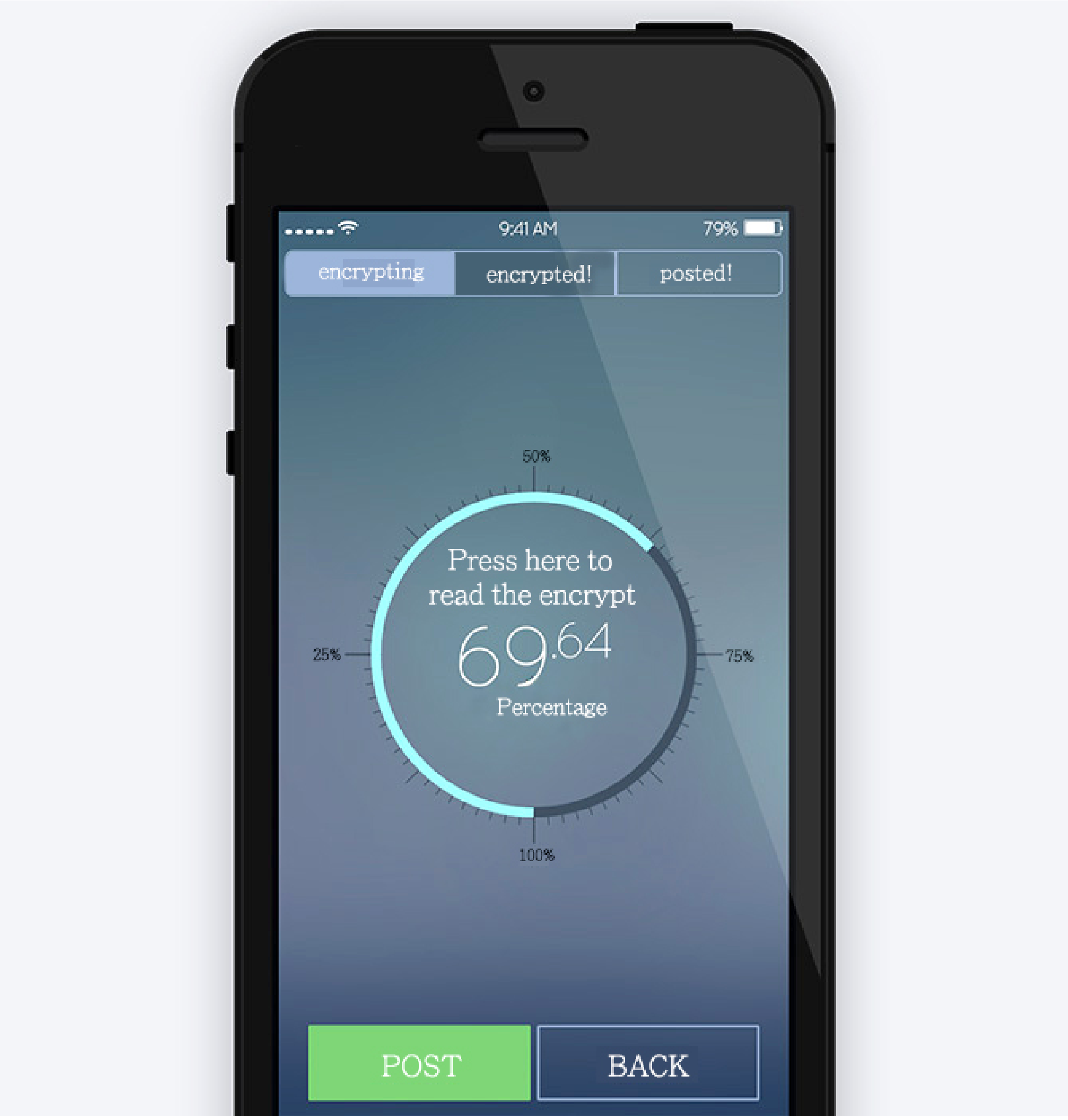


Figure 10.5: Mobile Sending Stage GUI

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.

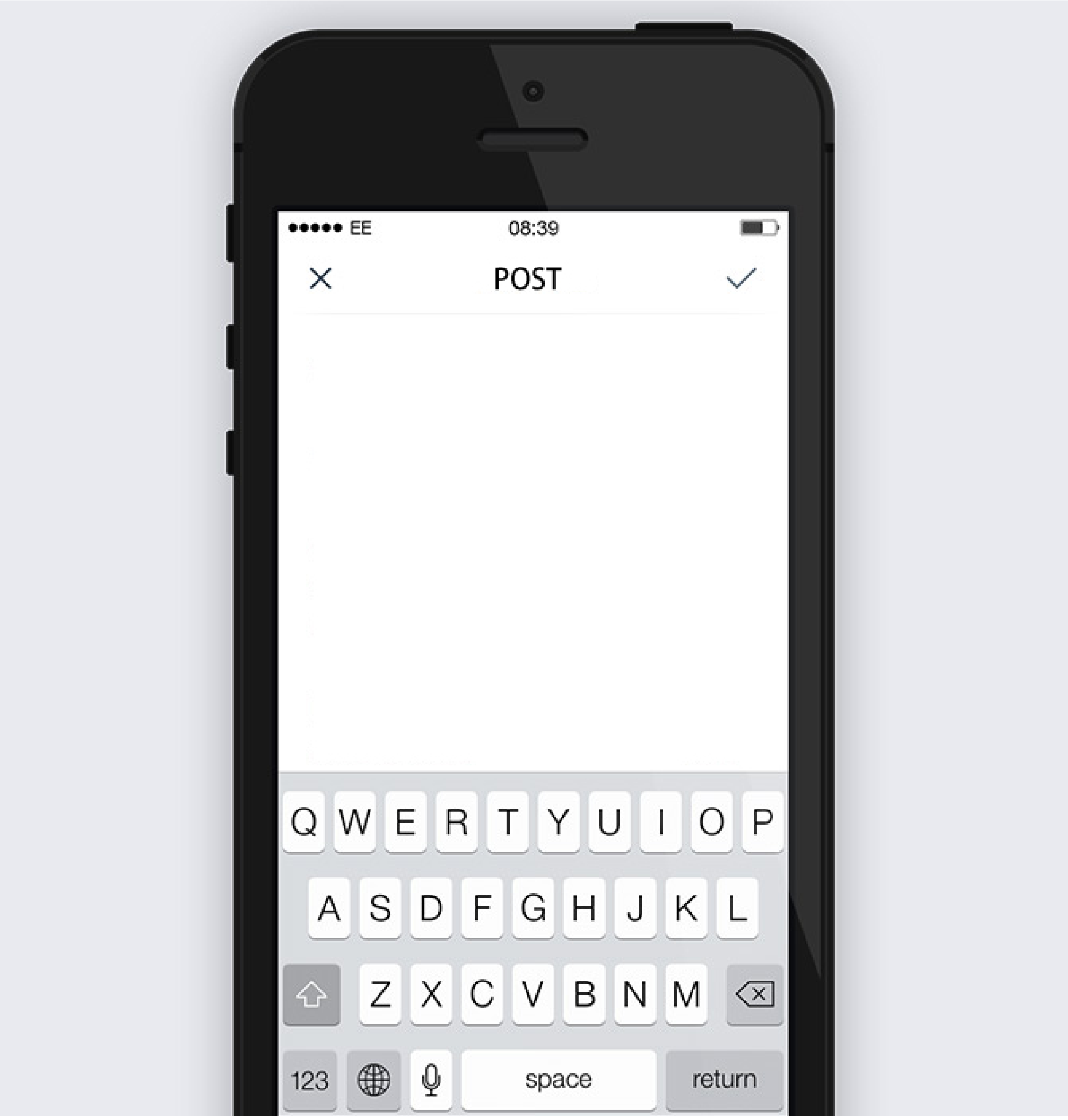


Figure 10.6: Mobile Post Stage GUI

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.

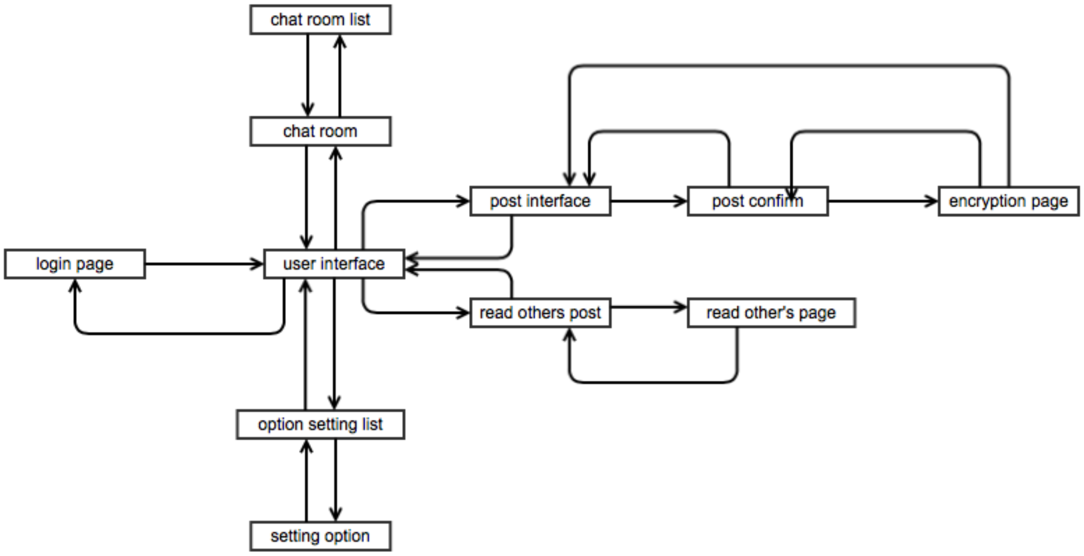


Figure 10.7: Mobile Data Flow Diagram

Chapter 11

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 12

QR

Another possibility of sharing our private keys with other users would be via the form of QR codes. One of our members located a website that generates QR Codes - both professionally and otherwise for free. It is possible to implement it into our program by having the program output the URL it gives us, as because it's generated via URL, we should be able to store it in a string and then output either that or have an image viewer in the program to output the actual image, whichever is easiest for the user.

The website's create function: `http://goqr.me/api/doc/create-qr-code/`

The website's read function: `http://goqr.me/api/doc/read-qr-code/`

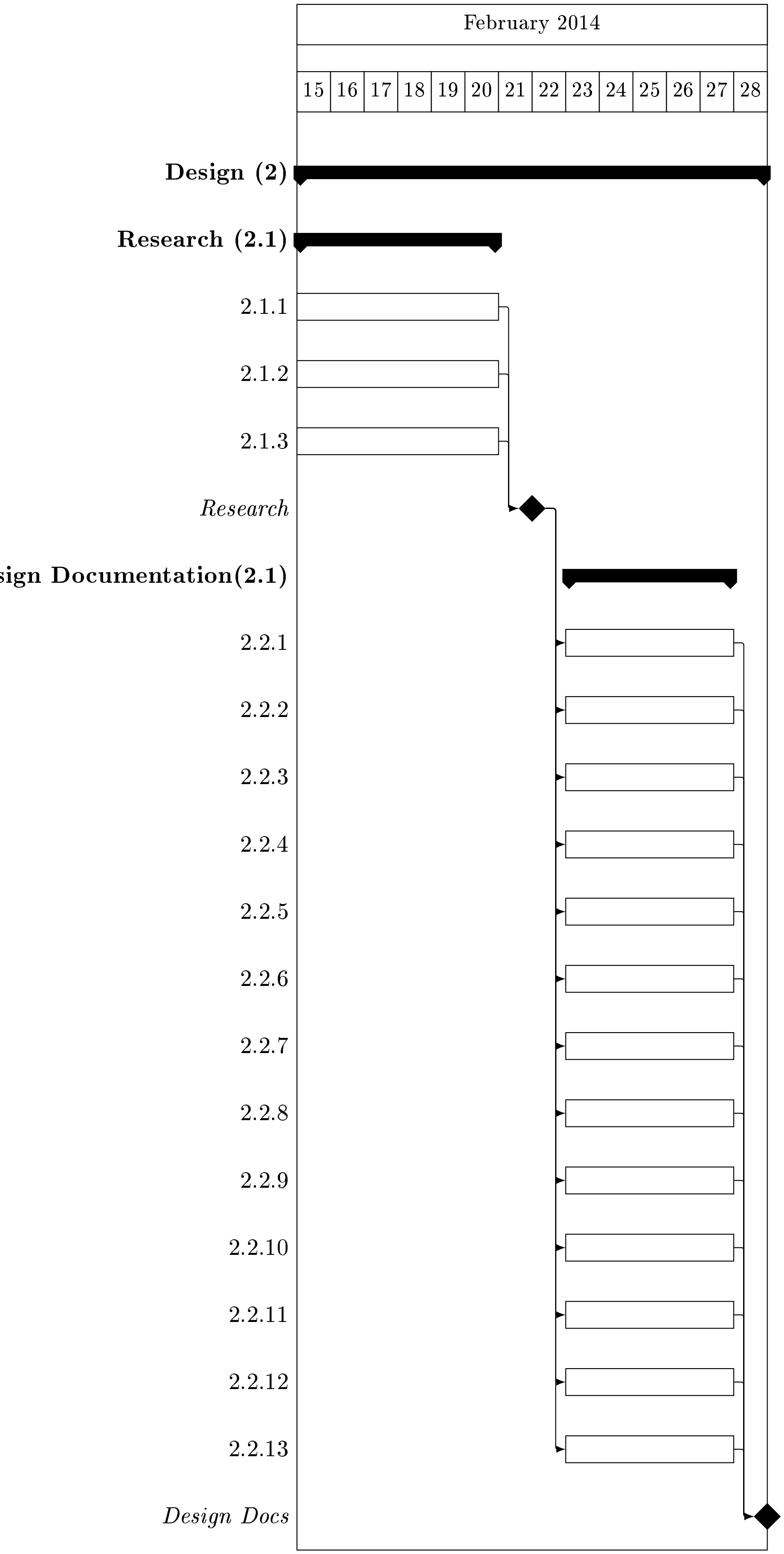
Test example using the website: `https://api.qrserver.com/v1/create-qr-code/?size=300x300\&data=\%3Ci'mThePublicKeyVariable\%3E\&format=svg`

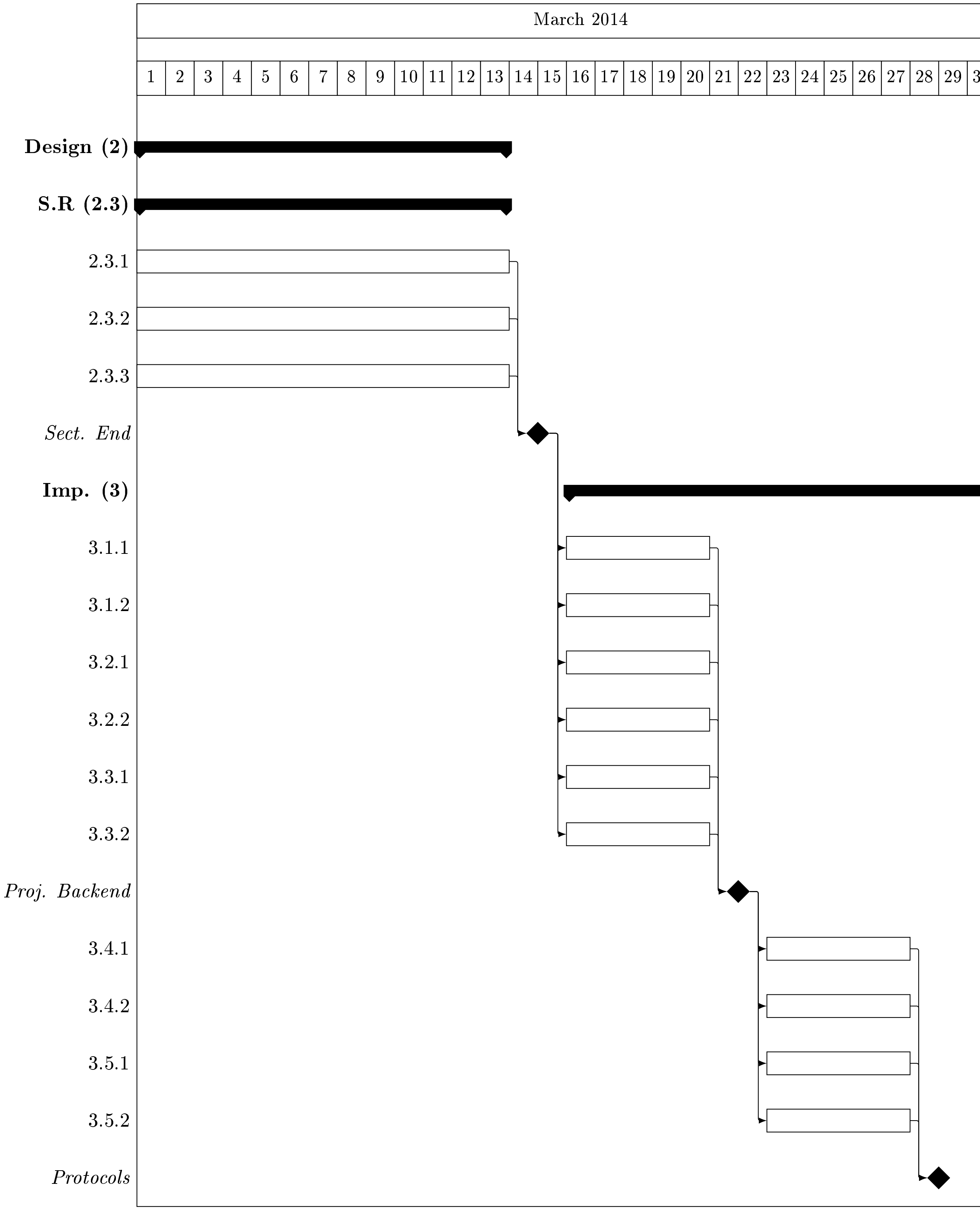
Chapter 13

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.


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 3
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

Chapter 14

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.

Appendices

Appendix A

Deadlines

- **2014-01-31** topic and team
- **2014-02-14** requirements
- **2014-03-14** design
- **2014-05-09** portfolio & individual submission

Appendix B

Licence

B.1 Statement of Purpose

The laws of most jurisdictions throughout the world automatically confer exclusive Copyright and Related Rights (defined below) upon the creator and subsequent owner(s) (each and all, an "owner") of an original work of authorship and/or a database (each, a "Work").

Certain owners wish to permanently relinquish those rights to a Work for the purpose of contributing to a commons of creative, cultural and scientific works ("Commons") that the public can reliably and without fear of later claims of infringement build upon, modify, incorporate in other works, reuse and redistribute as freely as possible in any form whatsoever and for any purposes, including without limitation commercial purposes. These owners may contribute to the Commons to promote the ideal of a free culture and the further production of creative, cultural and scientific works, or to gain reputation or greater distribution for their Work in part through the use and efforts of others.

For these and/or other purposes and motivations, and without any expectation of additional consideration or compensation, the person associating CC0 with a Work (the "Affirmer"), to the extent that he or she is an owner of Copyright and Related Rights in the Work, voluntarily elects to apply CC0 to the Work and publicly distribute the Work under its terms, with knowledge of his or her Copyright and Related Rights in the Work and the meaning and intended legal effect of CC0 on those rights.

B.2 Copyright and Related Rights

A Work made available under CC0 may be protected by copyright and related or neighboring rights ("Copyright and Related Rights"). Copyright and Related Rights include, but are not limited to, the following:

1. the right to reproduce, adapt, distribute, perform, display, communicate, and translate a Work;

2. moral rights retained by the original author(s) and/or performer(s);

3. publicity and privacy rights pertaining to a person's image or likeness depicted in a Work;

4. rights protecting against unfair competition in regards to a Work, subject to the limitations in paragraph 4(a), below;

5. rights protecting the extraction, dissemination, use and reuse of data in a Work;

6. database rights (such as those arising under Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, and under any national implementation thereof, including any amended or successor version of such directive); and

7. other similar, equivalent or corresponding rights throughout the world based on applicable law or treaty, and any national implementations thereof.

B.3 Waiver

To the greatest extent permitted by, but not in contravention of, applicable law, Affirmer hereby overtly, fully, permanently, irrevocably and unconditionally waives, abandons, and surrenders all of Affirmer's Copyright and Related Rights and associated claims and causes of action, whether now known or unknown (including existing as well as future claims and causes of action), in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty (including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever, including without limitation commercial, advertising or promotional purposes (the "Waiver"). Affirmer makes the Waiver for the benefit of each member of the public at large and to the detriment of Affirmer's heirs and successors, fully intending that such Waiver shall not be subject to revocation, rescission, cancellation, termination, or any other legal or equitable action to disrupt the quiet enjoyment of the Work by the public as contemplated by Affirmer's express Statement of Purpose.

B.4 Public License Fallback

Should any part of the Waiver for any reason be judged legally invalid or ineffective under applicable law, then the Waiver shall be preserved to the maximum extent permitted taking into account Affirmer's express Statement of Purpose. In addition, to the extent the Waiver is so judged Affirmer hereby grants to each affected person a royalty-free, non transferable, non sublicensable, non exclusive, irrevocable and unconditional license to exercise Affirmer's Copyright and Related Rights in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty (including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever, including without limitation commercial, advertising or promotional purposes (the "License"). The License shall be deemed effective as of the date CC0 was applied by Affirmer to the Work. Should any part of the License for any reason be judged legally invalid or ineffective under applicable law, such partial invalidity or ineffectiveness shall not invalidate the remainder of the License, and in such case Affirmer hereby affirms that he or she will not (i) exercise any of his or her remaining Copyright and Related Rights in the Work or (ii) assert any associated claims and causes of action with respect to the Work, in either case contrary to Affirmer's express Statement of Purpose.

B.5 Limitations and Disclaimers

1. No trademark or patent rights held by Affirmer are waived, abandoned, surrendered, licensed or otherwise affected by this document.

2. Affirmer offers the Work as-is and makes no representations or warranties of any kind concerning the Work, express, implied, statutory or otherwise, including without limitation warranties of title, merchantability, fitness for a particular purpose, non infringement, or the absence of latent or other defects, accuracy, or the present or absence of errors, whether or not discoverable, all to the greatest extent permissible under applicable law.

3. Affirmer disclaims responsibility for clearing rights of other persons that may apply to the Work or any use thereof, including without limitation any person's Copyright and Related Rights in the Work. Further, Affirmer disclaims responsibility for obtaining any necessary consents, permissions or other rights required for any use of the Work.

4. Affirmer understands and acknowledges that Creative Commons is not a party to this document and has no duty or obligation with respect to this CC0 or use of the Work.

B.6 Included Works

We did not write or create the following:

- writeup/latex/tikz-uml.sty
- writeup/latex/todonotes.sty
- writeup/latex/ulem.sty
- writeup/images/appendicies/licence.png (CC0 licence logo)
- The CC0 licence text
- client/web_interface_mockup/jquery.js
- client/web_interface_mockup/turtles.ttf

Appendix C

TODO

C.1 General

Errors shouldn't just display a message, they should be properly handled Get a real DB
REVOKE claims and messages after a certain date if private key leaked
escape backslashes in message content
chang all references to ascii text to UTF-8 text

C.2 Requirements Weeks 1-3

1. Project Desc.

- **COMPLETE** Project being done for (Peter)
- **COMPLETE** Mission Statement (Luke)
- **COMPLETE** Mission Objective (Luke)
- **COMPLETE** Threat Model (Luke)

2. Statement of Deliverables

- **COMPLETE** Desc. of anticipated documentation (Luke)
- **COMPLETE** Desc. of anticipated software (Aishah)
- **COMPLETE** Desc. + Eval. of any anticipated experiments + blackbox (Louis)

- **COMPLETE** User view and requirements (Luke)
- **COMPLETE** System requirements (Luke)
- **COMPLETE** Transaction requirements (Aishah)

3. Project and Plan

- **COMPLETE** Facebook research (Leon)
- **COMPLETE** Case Study: Tor (Luke)
- **COMPLETE** Case Study: alt.anonymous.messages and mix networks (Luke)
- **COMPLETE** Case Study: PGP and E-Mail (Luke)
- **COMPLETE** Implementation Stage (Peter)
- **COMPLETE** Milestone Identification (Milestones can most easily be recognised as deliverables) (Mike)
- **COMPLETE** Gantt Chart (Mike)
- **COMPLETE** Risk Assessment (Mike)

4. Bibliography

- **COMPLETE** Bibliography framework (Luke)
- **COMPLETE** Add citations where relevent (Everyone, in their own sections)

C.3 Design Weeks 4-X

- **DRAFTED** Use Case Diagram (Mike)
- **COMPLETE** Glossary (Mike)
- **DRAFTED** Mobile GUI Design (Leon)
- **DRAFTED** Sequence Diagram (Leon)
- **DRAFTED** HTML GUI Design (Louis)
- **COMPLETE** DB Design (Aishah)
- **COMPLETE** Transaction Design (Aishah)

- **DRAFTED** Server GUI Design (Peter)
- **COMPLETE** Class Interfaces (Luke)
- **COMPLETE** Protocol (Luke)
- **COMPLETE** Architecture (Luke)
- **COMPLETE** Data Flow Diagrams (Luke)
- **COMPLETE** Pseudocode (Luke)
- **COMPLETE** Class Diagram (Luke)