1 Introduction

Facebook, at the time of writing, is the world's most popular social network service with over 500 million users active in the last 30 days [6]. Encrypted Facebook is a Firefox extension which permits the use of broadcast encryption when sharing certain content via the Facebook platform.

1.1 Background

Since its inception, Facebook has come under criticism in relation to online privacy [1]. As social netorks mimic real life interactions members are often willing to reveal more private infromation than they otherwise would, resulting in social networks accumulating a large repository of sensitive information [4]. There are a number of ways this information can then be exposed: users misconfiguring privacy settings, [1] malice, error or neglect on the part of the social network and its employees, [2] acquiescent disclosure to government authorities, and unlawful access through phishing scams and security exploits [2] [**fb-gov**] [7] [11].

Encrypting content ensures its secrecy in any eventuality, provided the encryption key itself is kept safe. Tools exist to perform encryption manually, some are even partly integrated into the Facebook user interface [**firegpg**]. However, studies have shown that solutions such as these (which require management of cryptographic keys) are not usable enough to provide effective security for most users, putting them beyond the reach of the masses [**johhny**]. In addition, interaction over social networks is typically one-to-many, unlike applications like e-mail which these tools are

[1] Facebook was recently forced to update its privacy controls due to growing pressure from the public and media [8].

[2] As was the case in 2010 when personally identifiable Facebook user details were sold to data brokers by third party developers [3].

designed for. Other solutions have been proposed but they either fail to fully protect the user's privacy, lack support for common operations or suffer from performance, useability and scaleability issues (see sections 1.3 and 2.1).

Attempts have been made to create new social networks which protect privacy by encrypting content and in some cases even decentralizing hosting of the entire platform [**pidder**] [**diaspora**]. Unfortunately, network externalities make it difficult for newcomers to compete with Facebook since the utility of a social network is intrinsically tied to the size of its userbase. [3]

The aim of this project is to provide accessible, enhanced privacy capabilities to existing Facebook users. By incorporating a broadcast encryption scheme we ensure that the most popular forms of content can only ever be deciphered by the intended set of trusted recipients. Impact on the usebility of Facebook's services is kept to a minimum, in particular by integrating most of the controls into the Facebook UI itself. By ensuring compatability with normal Facebook activity incremental deployment is possible, avoiding some of the problems of networks effects.

1.2 Limitations

Some objectives we consider outside the scope of the project:

- Protecting against middleperson interception of public keys, discussed in section XXX, or against malware.
- Securing the actual structure of the social graph (see section XXX).
- Ensuring the integrity, authenticity or non-repudiation of communications. In theory the public key scheme could be extended to do so but this would go beyond mere privacy control.
- Guaranteeing availability of content. Completely severing reliance on Facebook would be tantamount to builing a new network.
- Concealing the existence of content itself or concealing the fact that it is encrypted. This may be possible but comes at a cost - see

[3] Some suggest the value of a social network grows linearithmically, quadratically or even exponentially with the number of users [10] [5].

appendix XXX for a full discussion. Steganography is employed but not for this reason (see section XXX).

- Cross-platform and/or cross-browser support (see section XXX).
- Designing and implementing a security policy which comprehensively covers all aspects of the project's functionality (see section XXX).

1.3 Existing work

Many applications for encrypting online data exist, most in the form of browser extensions [**firegpg**] [**cryptfire**] [**textcrypt**]. There are also several applications which target Facebook specifically.

uProtect.it Client side JavaScript which inserts UI controls and decodes content. Content stored and encrypted on a third party server.
`http://uprotect.it/index`

FaceCloak Firefox extension which posts fake (but convincing content) to Facebook, using it as an index to the encrypted content on a third party server. Running your own server is possible, though only users on the same server can communicate.
`http://crysp.uwaterloo.ca/software/facecloak/`

flyByNight Content is submitted through a Facebook application, encrypted using client side JavaScript and passed via Facebook to a third party application server. Proxy re-encryption is used for sending to multiple reciepients.
`http://hatswitch.org/~nikita/`

NOYB Content is stored in plaintext but profiles are anonymised. The mapping from real-to-fake profile is known only to a user's friends. See appendix XXX.
`http://adresearch.mpi-sws.org/noyb.html`

2 Preparation

Here we discusses possible approaches to protecting privacy in social networks and how Encrypted Facebook relates to existing work. We briefly review Firefox extension development, the Facebook platform, broadcast encryption schemes and other security related concerns. We also look at the specific problems assosciated with encrypting images and possible solutions. Finally, we consider an appropriate testing strategy and software development methodology and derive a concrete set of requirements.

2.1 Possible approaches

Protecting privacy involves tradeoffs between security, useability and scaleability. We outline possible approaches using examples from the literature and existing solutions.

2.1.1 General

It is possible to preserve pricacy by encrypting or otherwise hiding the link between a user and their online identity, as with NOYB. However, this approach suffers from problems of inference control and inherently degrades the quality of service provided by the social network. [1] As such, we do not consider NOYB or this method further.

The alternative is to encyrpt the content shared a social networkd user, restricting access to only those who possess the appropriate key. This is the method emplyed by Encrypted Facebook and the other proposals introduced in section XXX.

[1] See appendix XXX for a full discussion.

2.1.2 Outsourcing content

As well as encryption, FaceCloak, flyByNight and uProtect.it also opt to migrate content from the Facebook platform to an external database. There are some advantages to this approach: since very little is being stored on Facebook itself measures can be taken to hide the fact that content is being encrypted at all (text steganography is otherwise difficult due to the low amount of redundancy) [**facecloak**]. Better guarantees of availability can also be made in the case of account deletion.

Encrypted Facebook avoids outsourcing content due to scaleability concerns. Storing and delivering encrypted content requires at least the resources needed for storing and delivering the cleartext. Facebook is able to offer a free service by serving highly targetting advertising to its members based on the structure of the social graph [**fb-ads**]. This revenue stream would be largely unavaible to any solution hosting a database of encrypted content, and paying for privacy is unlikely to be an option for the majority of users.

2.1.3 Supporting multiple recipients

Communication over social networks is typically one to many. There are several ways this can be achieved using encryption:

- **Share a single symmetric key with you and your friends.** Requires secure out-of-band distribution of the key. Key revocation and providing fine grained control over recipient groups are problematic.
- **Keep your encryption key secret. Share your decryption key with your friends.** Subject to the same problems, however gives some protection against friends masquerading as you [**facecloak**]. A variant of this is used for FaceCloak.
- **Keep your decryption key secret. Submit your encryption key to a third party who performs proxy re-encryption.** Used by flyByNight. Re-encryption allows content encrypted under one keypair to be decrypted by another, without the third party ever being able to read the clear text [**flybynight**].
- **Authenticate with a trusted third party who posess all keys.** Confidentiality is only weakly assured: trust has simply been defered from Facebook to the third party and many of the scenarious raised in section 1.1 still apply. This does however greatly improve useability

by removing the need for users to manage any keys whatsoever and is the basis of uProtect.it [**uprotect**].

- **Use a more complicated broadcast encryption scheme.** Many schemes exist with various tradeoffs, discussed in section XXX.

Relying on a third party for encryption or re-encryption is subject to the same scaleability issue as outsourcing content. In addition, any method which requires out-of-band transmission will likely suffer from useability isssues relating to manual key management [**johhny**].

Encrypted Facebook uses a simple form of broadcast encryption, which is discussed in section XXX. No third party is required. Public key distribution is performed in-band; permitting semi-automated key management at the cost of vulnerability to middleperson attacks (discussed in section XXX). In addition, high resolution control over recipient groups and key revocation is possible.

2.1.4 Supporting images

Encrypted Facebook is currently the only solution that supports encrypted image content. Photo sharing is a primary concern for privacy concious users [**photos-priv**] and one of the most common activities (section XXX). Possible approaches are covered in detail in section XXX.

2.1.5 Intercepting Facebook interactions

In order to encrypt content it must be intercepted before being submitted to Facebook. There are several possible stages at which this may occur (Figure 2.1):

(a) A remotely hosted proxy server. Multiple users may access Facebook through the server.

(b) A proxy server running on localhost.

(c) Within the browser, outside the browser sandbox. Extensions and plugins operate here and have elevated priveledges over normal site code. Other examples include signed Java applets, ActiveX controls and to

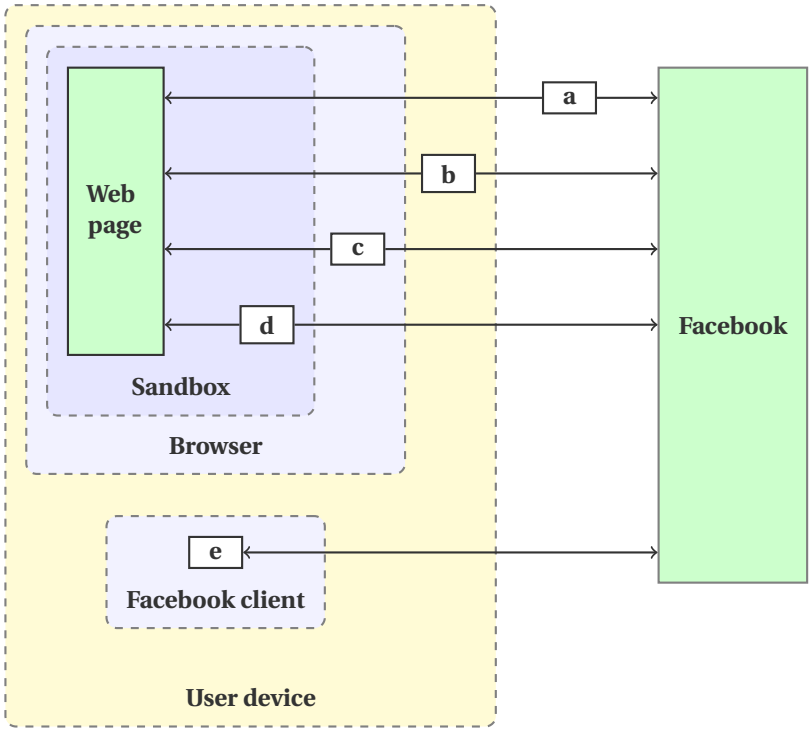

Figure 2.1: *Possible deployment strategys for intercepting interaction with Facebook.*

a lesser extent inline Flash and Silverlight applications. [2] FaceCloak takes this approach.

(d) Within the browser, entirely inside the browser sandbox - using only JavaScript and HTML. uProtect.it and flyByNight both take this approach.

(e) Outside the browser as a dedicated Facebook client application.

(a) isn't possible without relying on a third party for hosting. The browser sandbox prevents local filesystem access, ruling out (d) if private keys are to be kept locally. Encrypted Facebook takes approach (c) since (b) and (e) are considerably more complex, at some cost to cross-platform compatability.

Manipulating the **DOM!** (**DOM!**) is well supported by browser extensions since the browser interface chrome is often built on existing web technologies. This allows Encrypted Facebook to integrate functionality into Facebook's own user interface, which, along with distributing public keys in-band, mitigates usebility issues concerning key management.

2.2 Mozilla Firefox extension development

Encrypted Facebook is developed for Mozilla Firefox. Firefox is the worlds most popular browser (as of January 2011) and has a large support base for extension development [**ffox**]. Porting to other browser is discussed in section XXX.

Firefox extensions are written in JavaScript with partial support for Python and C/C++ [**ffox-lang**]. Performing cryptography in JavaScript is possible but comes with severe performance difficulties [**flybynight**]. Table 2.2 compares approximate performance for each language (full details given in appendix xxx).

Since long delays would hamper usebility Encrypted Facebook uses C/C++ for computation intensive operations. Native code can executed from within Firefox in at least three ways:

[2] Flash applications are restricted but can give basic filesystem access `http://blog.chromium.org/2010/12/rolling-out-sandbox-for-adobe-flash.html`.

Language	Library	Time (ms)
Python 2.6.6	pycryptopp 0.5.17	1,220 ms
C++ 98	Botan 1.8.11	92 ms
JavaScript 1.6 (in Chrome 12.0.712)	JavaScrypt (last updated December 2005)	1,685,000 ms

Figure 2.2: *Approximate time for 256-bit AES encryption of 1000 1.5 MiB random messages.*

- Creating an XPCOM component. These are linked against a single Gecko [3] SDK version; supporting multiple versions is possible but non-trivial [**xpcom**].
- Loading native libraries with `js-ctypes`. Introduced in Gecko 2.0 [**js-ctypes**].
- Using `nsiProcess` to invoke an external application. Capturing output can be difficult.

To maximise longevity Encrypted Facebook is designed to work on the latest version of the Firefox web browser, Firefox 4.0, which runs on Gecko 2.0. Since building an entire XCPOM component would be excessive and give little advantage by way of backwards compatibility, the newly introduced `js-ctypes` module is used to load native code.

Gecko 2.0 also provides better support for working with the local filesystem and manipulating images in the **DOM!**.

2.3 The Facebook platform

Encrypted Facebook supports encyrption for only the most popular types of object, listed in order of frequency of submission in Table 2.1.

Encryption adds size overheads. Without relying on a third party server all overheads must be stored on Facebook itelf, in some form or another. Images and blog-style notes are obvious targets for storage utilisation due to their high capacity limits (see Table 2.1). In particular, notes can contain

[3] Gecko is the layout engine used by Firefox.

over 120 KiB of information since each character represents one 16-bit unicode code point. Images are subject to lossy compression which is discussed in section XXX.

Activity	**Frequency (per second)**	**Limitations**	**Notes**
Comment	8,507	8,000 chars.	
Message	2,263	10,000 chars.	
Image	2,263	720 × 720 pixels	3-channel 8-bit colour. JPEG compressed (see section XXX).
Friend request	1,643		Social graph structure.
Status update	1,320	420 chars.	
Wall post	1,323	1,000 chars.	
Event invite	1,237		Social graph structure.
Photo tag	1,103		Social graph structure.
Link	833		
Like	unknown		Social graph structure.
Note	unknown	65,536 chars.	Used for blog-style posts.

Table 2.1: *Facebook objects, their limitations and approximate frequency of creation [9]*

2.3.1 Incremental deployment

Need to allow for INCREMENTAL DEPLOYMENT. Needs to be an opt in scheme, don't want to divide SNS. Non-users must be able to co-exist with users. By this we basically mean low signal to noise ratio, link somewhere about this. This might require deleting objects

2.3.2 Graph API

Now we know what we need to interact with, how do we do it? Facebook Graph API (JavaScript API etc.). What we can do. What we can't that we need to (writing to profiles, problems with album ids etc.) Why I didn't use the JavaScript SDK - poorly documented, working with images is a pain. For the workarounds - iframes and forms - see the implementation.

2.3.3 Connectedness

- Facebook says average is 130. This paper `http://arxiv.org/PS_cache/cs/pdf/0611/0611137v1.pdf` says average is 170 and distrubtion drops sharply at 250. Cameron Marlow says we only speak to a core group of friends anyway.
- Theoretical limits - Dunbar number of 150, others suggest higher at 300. So it perhaps won't increase with Facebook's expansion.
- Conclusion - 400 if we can make it work, but anything as low as 100 we could probably get away with.

Conclusion: we must be able to send objects X,Y and Z, and will probably need to make use of A,B and C. Three different ways needed to connect to Facebook to achieve this. No JavaScript SDK; must not significantly increase signal to noise ratio; must suport at least 400 friends/recipients.

2.4 Storing data in images

Images are the big thing. uProtect.it don't do images, FlyByNight talk about it as an extension.

- Description of Facebook/JPEG compression process (problem statement)
- Analysis of theoretical capacity
- Evaluation of naive implementations (MATLAB demonstrations). Obiously use gray codes to begin with.
  - Completely naive (encode in RGB values)
  - Slightly less so (encode in DST with bit masks)
  - Possible better schemes: Haar and Scale.

  - Optimal scheme? Dirty paper coding or similair? Its just too hard a problem but future work could be interesting.

- Conclusion: possible approaches : Haar and Scale. Modularity and extensibility useful though since we don't know which is best and both are sub-optimal.

2.5 Encryption schemes

- Threat model/analysis.
- Proxy re-encryption, like FlyByNite. Requires server side encryption so we leave it.
- Broadcast encryption.
  - Naive implementation.
  - More advanced schemes (and why I don't use them). Mainly due to no server side code, can't ask users to perform operations and expect a reply any time soon (or at all). We could reuse headers which would reduce a lot of size. In many ways this is a great idea, have a big linked list of notes containing links to headers, move frequently accessed entries to the front etc. However this means reusing message keys which is bad practice/NIST recommends against. Also, since we can only add and delete (not edit) notes (not through the API anyway, how fucking annoying) we might run into performance issues.
  - Underlying symmetric/asymmetric schemes. Maybe I could have improved the block size but ECC patents mean its not really found in open libraries.
- Conclusion : simple broadcast encryption using AES and RSA underneath. However, modularity and extensibility would be useful because there are improvements. In particular ECC would give dramatically smaller overheads.

2.6 Further security considerations

- Key management and size (NIST recommendations).

- Message key and IVs, don't reuse. Ensure good source of entrophy.
- Private key policy. Find a good reference, but basically we just mimic SSH and the like.
- Public key policy. Good idea to warn the user of the risks when they add public keys, check SSL is enabled etc.

2.7 Testing plan

- What kinds of testing will I use?
  - Unit testing , anything else??
  - Cognitive walkthrough - does this count as usability testing?
  - Security testing, since potential for exploits and project is security based - important enough to warrant its own section.
- How can I make these tests possible? Test bed or framework that needs to be in place?
  - Need a method of simulating the Facebook JPEG compression process. Use libjpeg since it most closely matches the compression signature (table of compression signatures). Show coefficient table.
  - Need a BER (bit error rate) calculator. Again coded as a C function.
  - FireBug and FireUnit for unit testing and profiling JavaScript functions.
  - gprof for profiling C/C++ functions.

2.8 Security testing

Loosely based on methodology here `http://mtc-m18.sid.inpe.br/col/sid.inpe.br/ePrint%4080/2006/12.20.12.15/doc/v1.pdf`. Must compromise since full security audit beyond scope of project. Look only at text retrieval process and public key management. We ignore images, and general attacks (e.g. setting up a spoof Facebook site). We also ignore threats that would be present ANYWAY e.g. if you haven't got SSL on. As an extension expand threat model.

- Threat analysis. Threat = Agent x Mechanism x Asset.
  - Facebook user creates a tag, which when decryption is attempted, causes denial of service (by locking up resources).
  - Facebook user creates a tag which when decrypted injects script in to page, gains control of users browser, can exectute arbitraty scripts within the Facebook domain (XSS) gains access to Facebook cookies.
  - Facebook user exploits UTF8 encoder/decoder to smuggle illegal characters past sanitization, gains control of users browser, can exectute arbitraty scripts within the Facebook domain (XSS) gains access to Facebook cookies.
  - Facebook user injects text which is run by JavaScripts eval() function, can execute arbitrary JavaScript outside the sandbox. Very Very bad!
  - Facebook user creates public key which, when parsed, creates a malicious file on the users local system.
- Risk analysis. Risk = (Vulnrability x Threat x Impact) / Security Measures.
  - Highest impact is running code outside the sandbox. True it maybe unlikely so long as we aren't stupid, but still. Basically we ban use of the eval function except for when we need it (retrieving JSON objects) then we replace it by a secureEval() which only allows valid Facebook object things.
  - Access to Facebook cookies can impact our security guarantees (since they could then change the public key). Also vulnrability is high. Thus we take time to sanitize before we inject into the browser.
  - Denial of service is low impact, but high vulnrability since the user need not do anyting to initate the decoding process other browsing to a site with a malicious post. So, test UTF8 decoder a lot, ensure that UTF8 decode, FEC decode, decrypt, all fail gracefully. Not image decode since out of scope, as mentioned above.
  - Public keys we can limit to Base64 characters of a certain length. Done.
- Test plan elabouration. From the above we want:

  - Testing of secureEval. Overide or otherwise ban eval().
  - Testing of text sanitiser.
  - Testing of UTF8 de/codec. Complicated given the large range of i/o.
  - Testing of public key downloader.

2.9 Proffesional practice stuff

- Software development methodology. Iterative prototyping. Work plan spells out which prototypes with what functionality should be completed when.
- Coding conventions, const correctedness etc.
- Version/source control. Git and project locker.
- Performance bounds. Of what???

2.10 Requirements analysis

- Encryption should be available on the most commonly used tasks (apart from those otherwise ruled out in section XXX). The user must therefore be able to broadcast-encrypt, submit, retrieve and decipher the following objects.
  - Status updates
  - Wall posts
  - Comments
  - Messages
  - Images

  Specifically, encryption should ensure confidentiality of data with at least 128 bits of security.
- All requirements should be met with recipient groups of size up to 400, which is a reasonable number - refer to discussion.
- Should be unobtrusive (refer to introduction) i.e. must not negatively affect browsing/Facebook experience of users. From this we derive the following:

  - Should try not to introduce any security holes. Up to a point, given scope of project. We have already declared a threat model and testing strategy etc.
  - Retrieval and submission times should be within acceptable limits. Define acceptable as `http://www.useit.com/papers/responsetime.html`.
  - Must not confine users to one computer. Should be portable. Securely transporting private keys is up to the user however.

- User activity should not negatively affect the activity of non-users (because of XXX refer to rest of preparation). We know there has to be some increase due to, for example, broadcast encryption overhead and status update's tiny length. Lets say maximum of twice number of objects generated compared to a normal user for the same activity.

- There are uncertainties and/or tradoffs associated with certain approaches to encryption and encoding data in images (and to a lesser extent error correction). It is also clear that in some cases the optimal approach is well beyond the scope of this project. Therefore, it is highly important that we adopt a modular structure that fascillitates switching between differing schemes and permits future extension. This need not extend to simultaneously supporting different schemes - this would introduce much redundant complexity.