Private Tweeting

Bas Doorn
851634462
Leiden
Netherlands

Lucas Vos
838820696
Hendrik-Ido-Ambacht
Netherlands

May 31, 2017

Begeleider: Hugo Jonker
Examinator: Tanja Vos
T61327 - Afstudeerproject bachelor informatica
Open Universiteit Nederland, faculteit Informatica

Abstract

Online social networks have become an increasingly popular phenomenon in the last decade. However, users tend to overlook the consequences of sharing personal information on these networks. Some operators allow users to hide certain information, but almost never from the operator himself and often only content information. We propose two solutions that enable Twitter users to hide their social network from Twitter. To do so, we first operationalise the concept 'social network' and various relevant privacy concepts. We then introduce the two models that we have developed and present the design and operation for both the relation establishment and the tweeting process. We also informally analyse to what extent they realise privacy of the social network. The first model is based on creating fake accounts that are used to communicate exclusively with each relation. This is called the 'accounts-model' and although it is not implemented, it is formally verified in the applied pi calculus. The second model uses a shared and public Twitter account that relays tweets to the recipients. This model hides the social relation by allowing users anonymously read tweets. We have analysed different cryptographic solutions to realise an efficient and secure way to communicate for this model, which also has been implemented as a proof-of-concept.

Contents

Chapter 1

Introduction

Social media have become an enormously popular phenomenon in the last decade. According to 'Nationale Social Media Monitor 2016', the Netherlands has 9.6 million Facebook users, 4.2 million LinkedIn users and 2.6 million Twitter users. The platform with the most users in the Netherlands is WhatsApp, with 9.8 million users [TKvdV15]. There is a wide variety in types of social media, which all have their particular functionalities and characteristics. These vary from very generic platforms like Facebook and Whatsapp to niche sites such as LinkedIn, focusing on professional networking [KHMS11]. A general definition of a social network was provided in Boyd and Ellison [CGC11]:

> We define social network sites as web-based services that allow individuals to (1) construct a public or semi-public profile within a bounded system, (2) articulate a list of other users with whom they share a connection, and (3) view and traverse their list of connections and those made by others within the system.

According to this definition, a public or semi-public profile is an essential aspect of an online social network (OSN). Also, the ability to view and traverse through the connections of another user is defined as a fundamental aspect of social media platforms. This is endorsed by Kietzmann et al, who define seven functional building blocks of social media. Four of these are *identity, sharing, presence* and *relationships* [KHMS11]. The emphasis of these platforms on sharing raises questions about privacy. Even though the user chooses what he or she shares, a lot of personal information is being made public. Even though users of OSNs are becoming increasingly aware of this, it is hard for them to fully gauge the impact of their disclosures. Meanwhile, the social network operator (SNO) manipulates users to disclose more and more because their business model often benefits from users sharing as much information as possible, either with the SNO itself, the public or both [BJE+12]. The maxim "if you're not paying, you're the product" certainly applies to this situation. The SNO is thus not acting in the user's best interest as a matter of course.

While it may seem contradictory at first sight, privacy on OSNs is a reasonable demand. Especially Twitter is often used as a platform for protesters and activists in totalitarian regimes, which has sometimes been called 'the Twitter Revolution' [BS11]. Although the actual effects of Twitter have been criticised as much as they have been praised (see for example [Mor11]), it cannot be denied that Twitter did play a role in the revolutions and protests that were called the Twitter Revolution. While leaving the discussion whether OSNs have a positive effect on these events to other experts, it reinforces the need for the ability to communicate ideas anonymously

or at least pseudonymously. While it might be easy for users to choose a username that is not easily linked to the person, there are other possibilities to link a pseudonym to an identity. For example, it has been proven that someone's social network is rather unique for a person. This can be used to identify a user in a set of other users [NS09]. While the SNO often provides some features that allow the user to hide certain aspects, including a user's social network itself, this almost never does not hide this information to the SNO itself, which is a problem when the SNO cannot be trusted. People like Edward Snowden or Julian Assange might very well have had more followers on Twitter if everyone could be sure that the US intelligence agencies would not be able to determine whether this kind of information is shared with them by the SNO.

As will be further elaborated in chapter 2, limited research has been conducted on how to hide the social network from the SNO itself. Therefore, this thesis aims to answer the following research question:

> To what extent is it possible to hide the user's social network from the social network operator?

In chapter 2, the notion of privacy on OSNs will be further elaborated. This chapter furthermore includes the contribution of this research, an explanation of the OSN Twitter and an overview of the literature that already has been conducted in this field. After that, the design of the research will be described in chapter 3, including two different designs and a description of an implementation of one of the models. In chapter 4, the different notions of privacy and anonymity will be more precisely defined, which will serve as a framework for a modelling the privacy of the social network in the applied pi calculus. An exploration of different cryptographic solutions for one of the models will be given in chapter 5. Finally, some conclusions will be drawn in chapter 6.

Chapter 2

Research context

2.1 Privacy in online social networks

Since privacy of the social network is a somewhat fuzzy concept, we will operationalise the research question in this section. First, we will sketch privacy concepts in the social networking environment. Then, we will provide a definition of the social network and social relations. After that, we will introduce some relevant privacy concepts. Most of the concepts are only briefly explained in this section, but will be explained in more detail and more formal way in chapter 4. While this section should provide sufficient information to be able to understand what the models that are proposed in the chapter 3 accomplish, it may be worth reading section 4.2 if any unclarities arise.

2.1.1 The social networking environment

From a privacy perspective, it is important to distinguish two different relationships: the relationship between user and the public, and the relationship between the user and the SNO. As mentioned in the introduction, there is a discrepancy between the interests of the SNO and the user. This implies that they must reach a compromise, which manifests itself in the extent to which the user is allowed to set the privacy preferences by the SNO. The introduction of 'protected tweets' on Twitter is an example of such a compromise. However, these compromises often only concern what users shares with the public, not what the users share with the SNO. Examples of implementations where the SNO makes certain aspects of the OSN inaccessible for itself are much rarer, with the recent update on WhatsApp which implemented end-to-end encryption for all chats as a notable exception.

Next to the question from *whom* a user would like to hide a certain information aspect, it is also relevant *what* information aspects needs be hidden. Beye et al define nine different aspect of an OSN that can be privacy-sensitive: profiles, connection, messages, multi-media, tags, preferences/ratings, groups, behavioural information and login credentials [BJE+12]. For the purpose of this thesis we will simplify this list by classifying these into three main categories:

1. *Content*: profiles, messages, multimedia
2. *Social network*: connection, tags, groups
3. *Metadata*: behavioural information, login credentials, preferences

While an SNO generally provides some sort of privacy-protecting measures for each of these categories for hiding the information from the public or other users, the efforts that are made by the SNO to apply measures to hide information from the SNO itself are virtually absent. If there are any, it is often only for only one of categories mentioned above. For example, the implementation of end-to-end encryption in Whatsapp just applies to information in the first category: only messages and multimedia are hidden from the SNO. This classification will be used a guideline to analyse both privacy-protecting measures that have been put in place by the SNO itself and privacy-protecting measures that have been proposed or developed in other research.

2.1.2 Social network and social relations

A social network is often described as a social structure that is made up of a set of persons and dyadic ties between these persons. We will call a dyadic tie between two persons a 'social relation'. This implies that a social network is built up from social relations. The study of social networks is called 'social network analysis', which can be conducted on social networks of any size. For example, [Gra16] have conducted a network analysis on 2,500 Twitter users.

For the purpose of this thesis, we are actually interested in a specific social network, which we will call a 'person's social network'. A person's social network is a social network, where one person is a subject of all the social relations in the social network. For example, Bob's social network consists of all the social relations between Bob and the people he knows. For the rest of this thesis, when we mention a social network we are referring to this type of social network.

Since we are developing a privacy-enhancing layer for Twitter, we should note that a person's social network on Twitter is probably different from a real person's social network: you may follow people on Twitter that you don not know in real life and you may know people in real life that don not use Twitter. Nevertheless, there may be overlap between the two networks. For this reason, we will consider the 'social Twitter network' a proxy for a person's social network. Besides the fact that a social Twitter network may reveal information about a person's real social network, there may be other aspects to it that can be worth hiding as well, which will be touched upon in section 2.1.3.

While a social Twitter network also consists of social relations, these are slightly different from social relations in a person's social network. In a person's social network, we assume that if Alice knows Bob, Bob knows Alice as well. This means that all relations are bidirectional. While this type of relation may be used in social media as well (for example, Facebook), Twitter uses unidirectional relations, which means that if Alice follows Bob, this does not imply that Bob follows Alice as well.

Having summarised the different types of social networks, it should be somewhat clearer what exactly should be hidden: a person's social Twitter network, which is made up of the following-relations where the person is a follower or being followed in.

2.1.3 Privacy concepts

First of all, we will consider various scenarios why a user would want privacy of the social Twitter network. As pointed out in section 2.1.2, the existence of a following-relation may indicate the existence of a social relation between those users. This may be undesirable if a user wants

to keep his social network private. But the fact that Twitter uses unidirectional relationships raises the question whether there are use cases that are only relevant for one direction: hiding that you follow another user or hiding that you are followed by someone. The following three scenarios should give an example of why this may be desirable:

- *Using Twitter anonymously.* Alice wants to use Twitter anonymously. Even if she uses anonymisation techniques such as TOR to hide her IP-address and uses an anonymous email-address to register her Twitter account, she may be identified based on her social Twitter network using re-identification techniques as shown by [NS09].
- *Listening anonymously.* Bob is interested in Edward Snowden, but does not want express this interest to Twitter or third parties that may have access to Twitter's data. He thus wants to follow Edward Snowden without disclosing the fact that he follows him, but does not care about hiding who follows him.
- *Tweeting anonymously.* Charlie wants to broadcast messages to a set of users, but does not want to disclose who is interested in his ideas. He does not care about sharing who he is following.

The different use cases described above demonstrate the need for different types of privacy. Before we will introduce the three types of anonymity that accomplish the desired type of privacy, we have to introduce the concept 'unlinkability'. Unlinkability means that certain items of interest (IOIs) cannot be linked to other IOIs by an adversary [PH10]. These IOIs can for example be tweets or users.

- *Relationship anonymity.* An adversary cannot sufficiently determine whether a relationship between two subjects exists. In terms of unlinkability: the subject and receiver cannot be linked to each other and any messages that are sent between these subjects cannot be linked to sender and receiver at the same time [PH10].
- *Recipient anonymity.* To a potentially receiving subject, each message is unlinkable [PH10].
- *Sender anonymity.* To a potentially sending subject, each message is unlinkable [PH10].

Interestingly, recipient anonymity and sender anonymity both imply relationship anonymity, if these are always ensured [PH10]. More complex variations arise when sender anonymity and recipient anonymity alternate. These variations will be discussed in section 4.2.

2.2 Twitter

In order to narrow the scope of this research, we will focus on one particular OSN, namely Twitter. Twitter has been around since 2006 and is often classified as a 'microblogging' website. It allows users to send out 140-character messages called 'tweets'. Unlike Facebook, where a connection is always bidirectional ('friends'), Twitter only provides the possibility to have a unidirectional connections, which is called following. While Twitter only has unidirectional connections, it is possible to for two persons to follow each other. This is important from an analysis perspective, because it provides more information about users than if only bidirectional connections would exist. By default, tweets are public but only followers will automatically see that person's tweets in his or her feed. Twitter has implemented a functionality called 'Protected Tweets'. This gives the user the possibility to not only hide the content of his or her tweets, but also the user's followers and users he or she follows. However, the number of followers and

users he follows is still public[1]. Protected tweets gives the user the ability to hide both content and his social network from the public and other users, but not from the SNO. Furthermore, Twitter's privacy settings allow the user to choose whether the user can be tagged in photos and if the user can be found by searching on email-address and/or phone number. To our knowledge, these currently are the only privacy-protecting measures that are offered by Twitter. While these measures do a reasonable job in protecting content and social network information from the public and other users they offer no protection from the SNO itself. Because the SNO is not by definition trusted or trustworthy, this functionality is desirable [BJE+12].

2.2.1 Legal considerations

One of the concerns for this project is the question to what extent this approach is compliant with the terms and conditions that a user agrees with when using Twitter. There are several documents explaining terms of usage of Twitter, which are all combined in the *User Agreement*. The user agreement of Twitter consists of three parts:

1. Twitter Terms of Service[2]
2. Twitter Privacy policy[3]
3. Twitter Rules[4]

In the following subsections, these documents will be examined to analyse the extent to which the approach of the project is compliant with the terms and conditions of usage for Twitter.

Twitter Terms of Service

By using the services provided by Twitter, one agrees with Terms of Service. In this document, a list of prohibited activities is provided. The focus is mainly on general service rules like tampering, spamming, hacking, etc. One particular article requires further examination:

> You also agree not to misuse our Services, for example, by interfering with them or accessing them *using a method other than the interface* and the instructions that we provide.

Especially the emphasised part could be a concern, because one of the ideas to implement the hiding of the social network from Twitter itself, is to built an special interface on-top-of Twitter. However, we assume that the official Twitter API is considered part of the Twitter interface, and programming an interface on top of the Twitter API is therefore not a vialotation of the Terms of Service.

Twitter Privacy policy

The privacy policy of Twitter describes in detail which personal information, tweets, contact information and other information is used by Twitter, and in which way they are allowed to use this information. This document did not contain any terms that were deemed relevant to this project.

[1] See https://support.twitter.com/articles/14016.
[2] `https://twitter.com/tos`
[3] `https://twitter.com/privacy`
[4] `https://support.twitter.com/articles/18311`

Twitter Rules

As one of the largest social networks in the world, Twitter has a great responsibility to ensure that the content that is generated by the users are 'socially appropriate'. The rules mainly concern the content of tweets, unlawful use of the services, spam and abusive behaviour such as harassment and hateful conduct. None of the rules explicitly prohibited the goals of this project, but there are two rules that should be examined in more detail.

- "Multiple account abuse: creating multiple accounts with overlapping uses or in order to evade the temporary or permanent suspension of a separate account is not allowed."
- "Impersonation: you may not impersonate others through the Twitter service in a manner that is intended to or does mislead, confuse, or deceive others."

Both the models that will be described in chapter 3 use at least one account that is created to enable the privacy features this project aims to develop. The Twitter rules state that these accounts may not be used for either overlapping uses or to evade the temporary or permanent suspension of a separate account. While the second purpose is clearly not applicable in this case, the term 'overlapping uses' is debatable. Although the multiple accounts are used to hide information, they do not impersonate another user.

Conclusion

In this brief analysis of the various terms and conditions that Twitter imposes on its users, we found one clause that may be relevant for this project: *multiple account abuse*. However, the extent to which the accounts that are created in this project have 'overlapping uses' is unclear. As far as we know, the idea of hiding the social network from the SNO itself is a new concept. As such, no explicit ruling on this matter was expected. Furthermore, since this project is a research project that merely aims to prove that hiding the social network from the OSN is theoretically and practically feasible, we conclude that the terms and conditions are not an impediment for the implementation of this project.

2.3 Related work

In the academic literature, it has been frequently argued that there is a lack of anonymity and privacy in online social networks [BBS+09, CMS09a, TSGW09, SZF10, BMP11, ZSZF10]. While some of these studies already point that a user's social network might be just as privacy-sensitive as the content or other information, this was especially made clear by [NS09], who developed an algorithm that was able to re-identify verified Twitter users using the social network information anonymous Twitter graph. Re-identification is particularly relevant for Twitter, because Twitter does not enforce a real-name policy, which can be effectively voided using re-identification methods, as shown by [PRC14].

To tackle these shortcomings in existing OSNs, numerous privacy-protecting technologies have been proposed in academic literature, of which [BJE+12] have provided an excellent overview. However, their analysis is broader than what is relevant for this research. They have divided the existing research until 2012 into four categories:

1. *Anonymisation*: this mainly covers anonymisation of information which is sold to other parties by the OSN.

2. *Decentralisation*: this provides an overview of decentralised models to hide information from the OSN.

3. *Privacy settings and management*: this mainly concerns the abilities that are provided by the OSN to the user hide information from the public and/or other users.

4. *Encryption*: an overview of existing research on the implementation of encryption measures is given, which may provide confidentiality and integrity towards public, other users and the OSN.

We will briefly give an overview the most important privacy-protecting solutions for OSNs that have been proposed. For this paper, the studies that are illustrated in categories *decentralisation* and *encryption* are considered most relevant.

Various studies have proposed alternative OSNs that claim to offer more privacy and/or anonymity than traditional OSNs:

- Persona [BBS+09] is an distributed OSN where users can decide who can access their information by using a combination of attribute based encryption (ABE) and public-key cryptography.

- Safebook [CMS09b] is another distributed social network based on a decentralised peer-to-peer network. The user's privacy is ensured by capitalising real-life trust relations between users.

- Unlike most of the other proposed frameworks, [ADBS09] argue that a centralised model is preferred over a decentralised model. While anonymity is not within the scope of this study, a solution for hiding both content and social network information is provided.

- Backes et al provide a more general cryptographic framework to achieve access control, privacy of social relations, secrecy of resources and anonymity [BMP11]. This is provided in the forms of a security API, which could by used by any OSN but is especially useful for decentralised OSNs. The study includes a formal verification of the framework in the applied pi calculus.

- A similar approach has been chosen by [SZF10], who propose a privacy-preserving scheme for data sharing in OSNs.

- PrPi [SSN+10] aims to develop a decentralised social networking infrastructure, on which new social networking applications can run. The authors have developed a new language called 'SociaLite', which can be used to develop such applications.

While all of the proposed solutions preserve privacy much better than traditional OSNs and these approach provide greater flexibility in implementing a model that accomplishes the desired effect, we think that it is worth recognising that other aspects than privacy play an important role for users in the decision which OSN they use. These aspects include usability and what platform the user's friends already use. Moreover, it is difficult to compete with the popular platforms, because they have huge financial resources and a vast user base, which enables them to constantly improve their product. For these reasons, we believe that a more realistic approach is to develop privacy-protecting layer for an existing OSN, rather than an alternative platform.

A privacy-protecting layer for Facebook called 'FlyByNight' is proposed by [LB08]. FlyByNight is a Facebook application that uses client-side Javascript to encrypt and decrypt sensitive data. While this solution has similarities with the approach of this thesis, it focuses on hiding the content from the SNO instead of the social network.
An innovative approach that does take the sensitivity of the social network into account is Lockr [TSGW09]. Lockr decouples the social network information from other OSN functionality and uses a *witness hiding proof of knowledge (WHOPK) protocol* to prevent disclosure of the social network information.
An interesting approach is taken by Frientegrity [FBFF12], who focus on the integrity of the data on OSNs instead of the confidentiality. Their method enables users to detect 'provider equivocation'.

While most of studies mentioned above propose innovative solutions to increase privacy or anonymity in online social networking, they do not have a particular focus on hiding the social network from the SNO. The sensitivity of relationships between users was pointed out by [CFP07], who suggest to keep relationship data private with encrypted relationship certificates. Access to these certificates is granted based on distribution rules. As is pointed out by [DF07], this solution relies on a (trusted) central node. They propose a solution based on public-key cryptography, which was further improved in [DFVSGN08], where a new protocol was described that uses homomorphic encryption to overcome the shortcomings of the earlier mentioned studies.

To conclude this section on relevant work we briefly summarise the relevance of the literature that is presented above. First of all, many papers point out the privacy-sensitivity of information that is shared on OSNs, including social network information. Since our research question focuses on developing a privacy-enhancing layer for an existing OSN that aims to hide the social network from the SNO, these arguments are particularly relevant for this thesis as well. Furthermore, various solutions to these problems are proposed. Many of these studies propose alternative (centralised or decentralised) OSNs instead of a privacy-enhancing layer or focus on hiding other aspects such as content. Nevertheless, different aspects of these solutions were used in our approach as well. A simplified version of the access control lists based on public-key cryptography as proposed by Domingo-Ferrer [DF07] form the basis of our broadcast model.

2.4 Contribution

Studies on privacy in OSNs usually focus on improving the possibility to hide data of category 1. This study will primarily focus on hiding category 2 data from the SNO. While this may seem counterintuitive, there are various reasons to do so. First of all, the privacy-sensitivity of the OSN has been pointed out in various studies [NS09, PRC14, CX10]. Secondly, hiding the social network is more challenging than just hiding content data, because the social network can often be trivially reconstructed when content is not hidden. Hiding the social network therefore must build upon studies that have focused on hiding content information. To our knowledge, there are no examples of popular OSNs that implemented the possibility for users to hide the social network from the SNO itself. This is in contrast to the ability to hide the social network to some extent from other users or non-users, which is present in most of the OSNs[5].

[5]For example, enabling 'Protected Tweets' in Twitter effectively hides the followers and the persons being followed by a users. LinkedIn enables the user to choose who can see their connections {'only you', 'your connections'}. Facebook also provides the possibility to choose who can see the users' friend list.

Although there have been studies that consider hiding the social network from an OSN, these often propose solutions that entail an alternative (often decentralised) OSN [SSN+10, BBS+09, CMS09b]. While such an approach provides greater flexibility in implementing a model that accomplishes the desired effect, we think that it is worth recognising that other aspects than privacy play an important role for users in the decision which OSN they use. These aspects include usability and what platform the user's friends already use. Moreover, it is difficult to compete with the popular platforms, because they have huge financial resources and a vast user base, which enables them to constantly improve their product. For these reasons, the aim of this research is to develop a privacy-protecting layer for an existing OSN, rather than an alternative platform.

Chapter 3

Research design

3.1 How to hide the social network

As stated in chapter 1, we will research to what extent it is possible to hide the user's social network from the social network operator. In this chapter, we will try to answer that question by developing two models that effectively hide the social network by providing a privacy-enhancing layer for the popular microblogging platform Twitter. We will reference the privacy-enhancing layer generally as 'Pwitter'.
The first model will be referenced as the 'accounts-model'. It ensures sender anonymity by creating extra accounts that are used exclusively to communicate with one other user. Tweets are encrypted for each relation with symmetric key both participants share.
The second model is named 'broadcast-model' and realises recipient anonymity by using a shared broadcast account, which relays tweets. All tweets are encrypted using a hybrid cryptosystem and can only be decrypted by authorised users.

A detailed discussion of both of these models will be given in sections 3.2 and 3.3 and will encompass the following aspects:

1. *Introduction.* An informal description of the key concepts, methods and ideas behind the model.
2. *Design.* A schematic design of the model.
3. *Protocol description.* A more formal description of the protocols for relation establishment, tweeting and timeline construction.
4. *Scope of the afforded privacy.* A brief analysis of the scope of the afforded privacy.
5. *Limitations.* An overview of practical limitations of the model.

For practical reasons, only the broadcast-model is implemented as a proof-of-concept. The design will be elaborated in more detail in chapter 5, including an analysis of the underlying encryption mechanisms. A description of the actual implementation of this model will be given in appendix D. While the accounts-model is not implemented, it will be formally verified using the applied pi calculus in chapter 4.

3.1.1 Common concepts

Although the two models are quite different, they both have some common patterns and problems which will be discussed first in this section:

I *Users*. Different types of users and some general assumptions are introduced.

II *How to encode a tweet for private eyes only*. Possibilities to encode a tweet in efficient ways are discussed

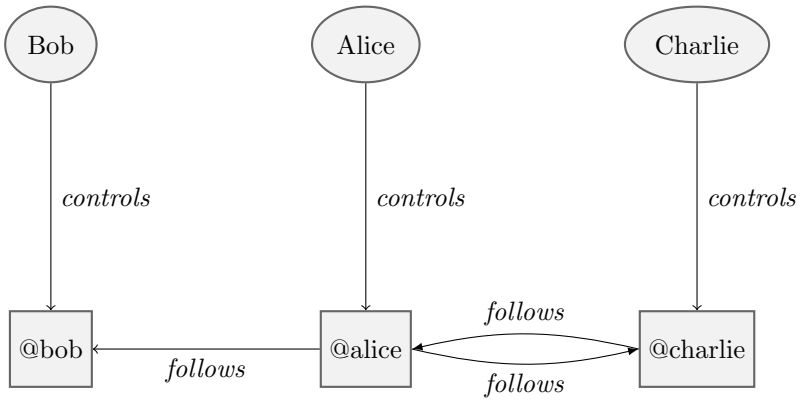
III *Data storage*. Possibilities to efficiently store both public and private data on Twitter are discussed.

I. Users

This application aims to implement a hybrid model where a smooth transition from normal users to Pwitter users is possible. This implies that any relationship between two users should not be affected if one of them starts using Pwitter. Thus, only part of Alice's social network will be hidden, which will increase as more users start using Pwitter. Unless stated otherwise, we will assume that all users involved use the 'Protected tweets' functionality of Twitter, which means that a user must authorise another user before the tweets are visible. The reason for this is twofold: first of all, we assume that users who value their privacy towards the SNO, also value their privacy towards the rest of the world and therefore will likely use protected tweets. But perhaps more importantly, the solution to the research question would be rather trivial if protected tweets would not be used: one could simply visit the pages of the users they would like to follow to see their tweets, which reduces the problem to a network anonymity problem, which can easily be resolved by using onion routing or other network anonymisation techniques.

In figure 3.1, a schematic overview of Twitter is presented, which will be primarily used as a reference for the schematic overviews of the accounts-model and the broadcast-model. The overview simply depicts three identities (Alice, Bob and Charlie), who all have a normal Twitter account, respectively @alice, @bob and @charlie. In this example, user @alice follows @bob and @charlie, and user @charlie follows @alice as well. User @bob follows no-one.

Figure 3.1: Twitter model schematic overview



II. How to encode a tweet for private eyes only

Since we assume a hybrid model, private tweets may be trivially linked to normal tweets based on the content of the tweet. To prevent this, the tweets can be encrypted. We will go into detail about the encryption in chapter 5, but regardless of the specific encryption methods, the ciphertext has to be embedded into Twitter's data transmission mechanisms. In this section, we will discuss various possibilities to encode the ciphertext.

Tweet encoding The most obvious possibility to is to encode the tweet data into a tweet-object itself. However, the 140-character limit for tweets[1] places an important constraint on the encoding and encryption mechanisms: the encoded ciphertext cannot not be longer than 140-characters as well. Since we do not want to have a lower character limit for Pwitter-users, the data needs to be encrypted, compressed and encoded as efficiently as possible, while the combination should still be secure. To realise this, the encryption scheme that is chosen needs to have as little ciphertext expansion as possible. The expansion may be dependent on various factors, such as the class of algorithm (symmetric or asymmetric), algorithm itself and mode of operation. For the rest of this section, we will assume that a symmetric algorithm is used. Secondly, compression can be used. Various algorithms exist, which can be benchmarked for this application. Finally, the biggest reduction in the amount of characters can be gained by encoding efficiently. Since Twitter supports Unicode, a mapping to Unicode characters is much more efficient than encoding with base64.

Direct messages A second possibility using Twitter's direct message system. These messages can be sent to other users and do not have the 140-character limit that tweets have, which makes it a good alternative to tweets. However, the viability of using the direct message system is dependent on the Pwitter model that is chosen. The accounts-model uses 1-to-1 accounts, so direct messages may be very well be used instead of tweets or images (see section 3.2 for a detailed explanation of the model). The broadcast-model (explained in section 3.3) however relies on broadcasting the message and the anonymity of the recipient. Since the recipient has to be explicitly chosen in direct messages, they do not fit the broadcast model particularly well.

Image encoding A third possibility is to encode the tweet into an image and attach the image to a tweet. When this method is used, the size of the content of the tweet is less important, but may contain some sort of header to allow the Pwitter client to efficiently determine whether it should try to decode the attached image or not. The data itself can be encoded in a newly generated image by setting the value of one or more channels in an image to 0-255, that correspond to one byte of a string. This results in a random image. Another possibility is to encode the message in a existing image, which obfuscates the fact that some extra information is embedded in the image. This can be considered a form of steganography. Altough the added security value of this is limited since it may be quite easy for the adversary to determine that Pwitter is used based on other indicators, it does not deny users to attach images to their tweets.

The advantage of this approach is that it allows for much more data, so the necessity of efficient encryption and encoding is less pressing. However, it could be considered less elegant than encoding the data into tweets, since an image needs to be created for each tweet.
Another disadvantage is that Twitter seems to apply some compression or other modifications to image when it is uploaded, which may result in loss of data.

[1] A detailed explanation of the exact limitations of the amount of bytes that a tweet can contain are given in the Twitter development documentation: https://dev.twitter.com/basics/counting-characters.

III. Data storage

In addition to the transmission of the tweets, there is some data that needs to be stored as well. This can be divided into two types of data: private data and public data. The data that should be kept private consists of data about the social network (users that you follow) and possibly private key material. The public data needs to be public for Pwitter to function. Dependent on the encryption scheme used, this may entail public key(s) and/or salt.

Local A first approach is to store the data locally. This way, the user is responsible for the data. However, there are some practical disadvantages to this approach. Most importantly, many users have multiple devices on which they use Twitter (laptops, smartphones, tablets). Only storing the information locally will result in synchronisation issues. Secondly, if users fail to backup the data they may lose access to the private data and unable to access the account and decrypt tweets.

Third party Most of the disadvantages of storing the data locally may be solved by storing it on a third party location (which may be controlled by Pwitter). The data can then easily be synchronised among various devices and is automatically backed up in case the data on the device gets erased or corrupted. Obviously, the data will need to be encrypted if a third party is used, otherwise is not an improvement over the original situation. The disadvantage of this approach is that extra infrastructure is needed, which needs to be maintained, paid and secured. This makes the application considerably less 'lightweight'.

The most efficient solution would be store the data somewhere within Twitter. This has the advantage that no extra infrastructure is needed and it would still be possible to use Pwitter across various devices. The disadvantage is that Twitter does not provide facilities to store general purpose data. This means we have to 'misuse' something to store the data. Some options will be discussed below.

Tweets It is possible to store the information in one or more tweets. Whenever logging into Pwitter, these specific tweets need to be fetched. This approach has various disadvantages. First of all, tweets are immutable. While the public data may remain constant, the private data will need to change quite often (every time a new follower is added or deleted). This implies that new tweets have to be posted after each change. Tweets can be looked up by their id, but the id is generated by Twitter, which means that after each change in the private data, a new reference for the tweets with private data has be kept somewhere. This shifts the problem instead of solving it. Furthermore, the timeline will get contaminated with tweets that will mean nothing to followers that not use Pwitter.

Direct messages A second possibility is that a user sends a direct message to himself. While some of the problems with referencing the message apply here is well, they should be easier to overcome as there are probably not many messages that a users sends to himself. If the conversation of a user with itself is reserved for Pwitter and new message is sent when the private data changes (the older ones can even be deleted then), it should be much easier to find the correct message. However, it also quite easy in the normal Twitter-interface to delete a conversation. That means it is quite risky to store the messages here, as all the private data could be deleted with one missed click.
An even more serious disadvantage is that the direct messages are not visible for anyone but the user itself, which means it is not possible to store any public data this way. That means this

method could solely be used for the private data and has be used in combination with another method to store the public data.

Images Similar to the tweets, data that needs to be stored can be encoded in the images that are part of the profile. A Twitter profile can have a custom banner, background or profile image. Embedding data in one of these images has the advantage that they can be easily referenced: the data is always in the same place and can be easily retrieved using both the Twitter API and the DOM-tree of the public profile page. While using randomly generated images denies the user to choose their own image, a steganography approach allows the user to encode the profile data within their image of choice. Encoding the data at the borders of the image may also reduce the impact on the image. Both the public and private data may be encoded here, where the private data will be encrypted.
The same disadvantage as with images attached to tweets applies here: Twitter seems to apply some compression that alters the data.

Conclusion

Having discussed various possible ways to transmit and store public and private data, we conclude that encoding the data in images is the best option. It allows to make the public data easily accessible and doesn't provide any practical limitations on the size of the message.

3.2 Accounts-model

3.2.1 Introduction

The first approach is based on 1-to-1 accounts: for each follower that uses Pwitter as well, an account will be created, which will be exclusively used to communicate with that particular follower. This means that if Alice has n followers that use Pwitter, $2n$ accounts will be created, of which n by Alice. When Alice broadcasts a tweet, all of these accounts will broadcast the same tweet. Since it should be unfeasible to link these accounts to Alice or to each other, the tweet will need be encrypted in such a way that the ciphertext is different for each account.

3.2.2 Design

In figure 3.2, a schematic overview of the account-model is given. As in figure 3.1, there are three identities (Alice, Bob and Charlie). For communicating with Bob, Alice uses her account '@alice_bob', which follow Bob's account '@bob_alice'. Similarly, Alice controls another account '@alice_charlie' that she uses to follow Charlie's account '@charlie_alice'. This account in return follows '@alice_charlie'. Note that none of the accounts in the overview follow more than one user.

3.2.3 Protocol description

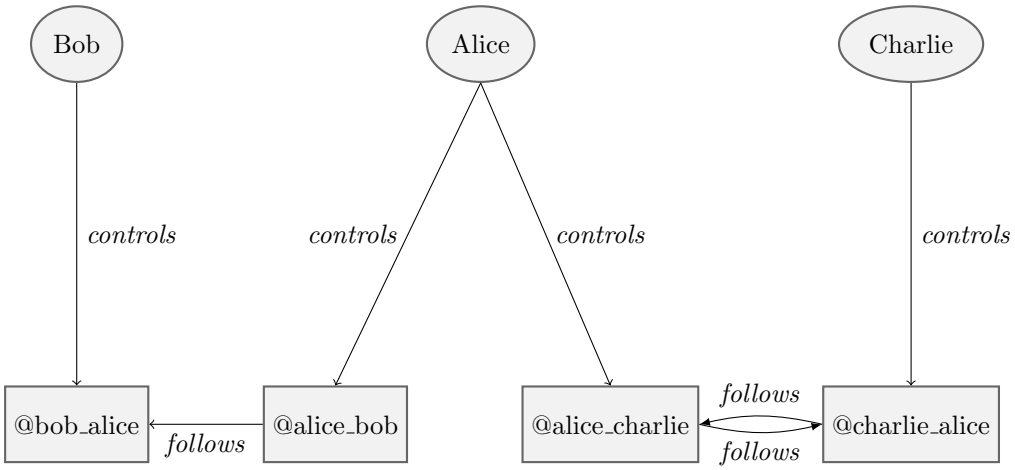
The design of the accounts-model can be divided into two phases:

I Establishment of the following relation (including key establishment).

II The tweeting process.

Each of these phases have their own particular characteristics, thus will be explained separately below.

Figure 3.2: Accounts model schematic overview



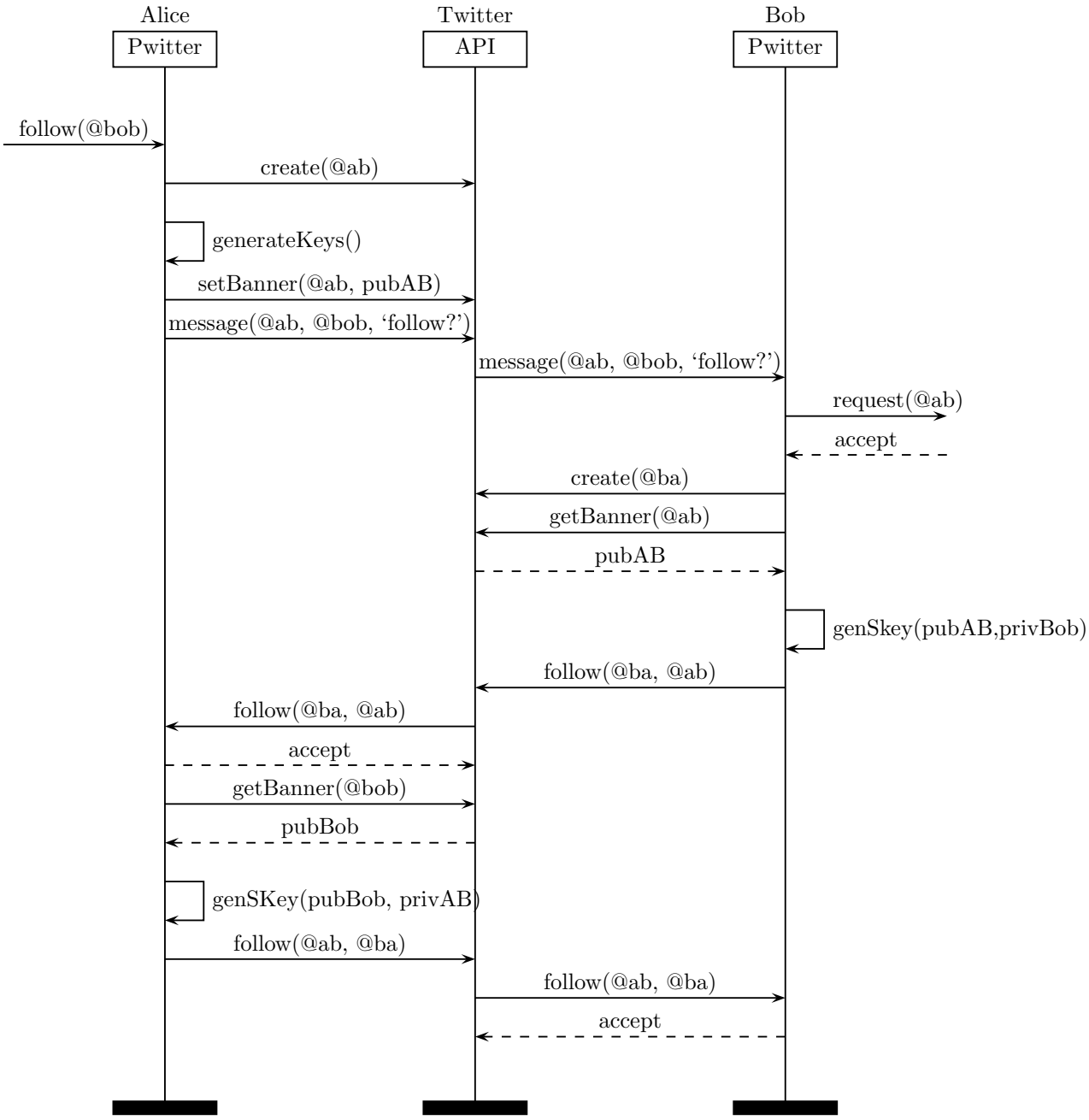
I. Relation establishment

The relationship establishment phase is arguably the most complicated part. It can roughly be divided into five sub-phases:

- *Fake account creation (Alice).* In this sub-phase, Alice creates an account *ab* and generates a public-key pair.
- *Following request.* Alice requests to follow Bob by sending a special message.
- *Fake account creation (Bob).* If Bob accepts he creates a fake account.
- *Twitter following requests (Bob).* Bob requests to follow Alice's fake account with his fake account. Alice accepts and requests to follow Bob's fake account with her fake account.
- *Key establishment.* Alice and Bob generate the symmetric key which they will use to encrypt their tweets.

The complete flow of the establishment of the following-relation is depicted in figure 3.3.

Figure 3.3: Relation establishment of the accounts-model



For the establishment of a following-relation between Alice and Bob we assume that each user that uses Pwitter has one static Diffie-Hellman (DH) key-pair, of which the public key is encoded in the banner-picture and thus retrievable from the profile page of the user. The fake account has a public key in the banner-picture as well, this is the ephemeral key. The ephemeral key and the static key are used to generate the symmetric key that will be used for encrypting the tweets between the two users. Each step in the protocol flow that is depicted in figure 3.3 is commented below:

- follow(@bob). Alice requests Pwitter to follow user '@bob'.
- create(@ab). Alice creates a fake account with random username @ab.
- generateKeys(). Alice generates a new pair of DH-keys: $pubAB$ and $privAB$. $PubAB$ is encoded in an image.
- setBanner($pubAB$). Alice and puts the image with public key $pubAB$ on the profile page of @ab.
- message(@ab, @bob, 'follow?') (1). Alice sends a direct message to user '@bob' with specific content.
- message(@ab, @bob, 'follow?') (2). The content of the message is recognised by Bob's Pwitter client as a friend request.
- request(@ab). The request is presented as a friend request from @ab to Bob in the Pwitter client.
- create(@ba). Bob accepts the request and creates a new account with username @ba.
- getBanner(@ab). Bob requests @ab's banner to obtain her ephemeral DH-key $pubAB$.
- genSkey($pubAB, privBob$). Bob can now compute the symmetric key using his own private key $privBob$ and @ab's public key $/mathitpubAB$.
- follow(@ab). Bob makes a following request from his new account @ba to @ab.
- request(@ba). Alice receives the following request and her Pwitter-client automatically accepts.
- getBanner(@bob). Alice requests @bob's banner to obtain his static DH-key $pubBob$.
- genSkey($pubBob, privAB$). Alice can now compute the symmetric key using her own private key $privAB$ and @bob's public key $pubBob$.
- follow(@ba). Alice now makes a following request from @ab to account @ba.
- request(@ab). Bob receives the following request and automatically accepts.

Both Alice and Bob now have each others newly generated usernames for the fake accounts. The communication on these fake accounts can be one-way (if Alice only wants to follow Bob or the other way around) or two-way (if Alice and Bob want to follow each other). In either case information is transmitted and received between the two fake accounts. Alice and Bob also both have a symmetric key k_{ab} which will be used to encrypt the tweets that are published on the fake accounts.

At first sight, it may seem unnecessary to create two fake accounts. However, consider the following scenario:

> Alice wants to follow Bob. Bob creates a fake account and Alice follows this account. Now Bob decides that he wants to follow as well, so Alice needs to create a fake account and Bob will need to follow this account. Alice and Bob now both have a fake account, but the fake accounts are being followed by the main accounts.

Since one-way relationship may very well lead to a two-way relationship, we should anticipate this by already creating two fake accounts in the first place. This means that if Alice wants to follow Bob (and Bob initially does not want to follow Alice), both Alice and Bob create a fake account. Only Alice's fake account follows Bob's fake account. If Bob at some point in the future wants to follow Alice as well, he can simply start following the fake account that is already following his own fake account.

II. Tweeting

The process of tweeting is depicted in figure 3.4. This is straightforward: the flow starts by Alice giving a message as input to the Pwitter client. Then, for each person that follows Alice, the client logs into the fake account @x that is used to communicate with that follower, encrypts the message with the symmetric key k_x for that user, tweets the encrypted message and logs out.

The construction of the timeline is shown in figure 3.5. When Alice calls the getTweets-function on the Pwitter client, it logs into each of the fake accounts that Alice uses to follow other users and fetches all tweets on these accounts. For each tweet per account, the client decrypts the content of the tweet using the symmetric key that is used for that following-relation (k_x). The loop ends with logging out. All decrypted tweets are then presented to Alice in the Pwitter client, sorted by time.

For this implementation will we assume the most basic model, where tweets can only be read when the application is opened. This means that it's not possible to get a push message or notification when a tweet is posted while the application is not open. When the application is opened, it needs to login into all different accounts. The n newest tweets per account will be parsed and decrypted, after which the timeline can be reconstructed. Note that this will happen every time the users logs in, so it's not necessary to store the decrypted tweets somewhere. Just as in Twitter, only the newest tweets will be loaded, but older tweets can be loaded as well by clicking on a 'load older tweets'-button.

Figure 3.4: Tweeting of accounts-model

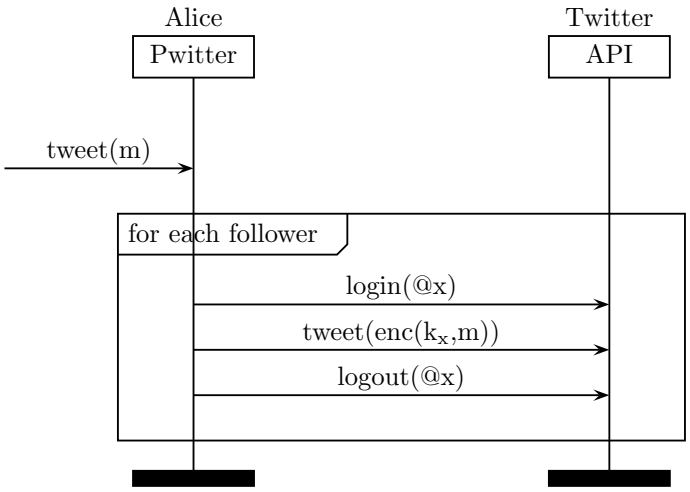
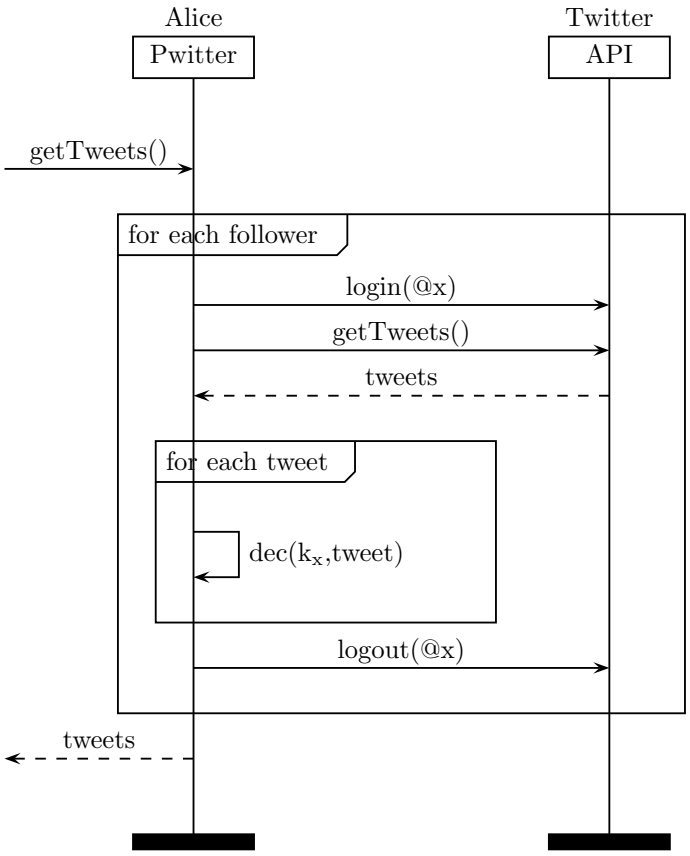


Figure 3.5: Constructing timeline of accounts-model



3.2.4 Scope of the afforded privacy

Although a formal verification of the privacy of the account-model will be given in chapter 4, it is desirable to informally analyse the extent to which this model successfully hides the social network from the SNO. We will first analyse the privacy of the model in terms of the concepts proposed in section 2.1. Then, we will present a brief risk analysis on the model, including mitigations for those risks.

Privacy of the accounts-model

When solely looking at the tweeting process, the model seems to ensure both sender and recipient anonymity. However, when looking at the relation establishment phase, one notices that recipient anonymity is broken by the following request that has to be made from Alice's fake account to Bob's main account. This means that the link between Bob's fake account and Bob is revealed to Twitter. The reason for still creating an extra account is mainly practical: otherwise Bob's other followers would receive encrypted tweets on their timeline if Bob sends a tweet. While it could be argued that this model ensures sender anonymity, the term is a little misleading in this case. The model ensures that Alice cannot be linked to her fake account, but she may both send a receive messages on this account. Technically, the model provides sender anonymity during the relation establishment phase and both sender and receiver anonymity in the tweeting process. Because sender anonymity implies relationship anonymity, privacy of the social relation between Alice and Bob is ensured.

Risk analysis

Linking tweets by content An obvious attack is to link tweets to other tweets based on the content.
This risk can be mitigated by encrypting the content of the tweet . Note that even if the message is encrypted it may be linked if the same key (and a deterministic algorithm) is used, because this will result in the same ciphertext. For these reasons, the account-model uses a different symmetric key for each relationship.

Linking tweets by timing Besides the content, other aspects may be used to link messages together as well. One of these is the timestamp of the tweet. If n accounts send a tweet on the same time or shortly after each other, the SNO may use statistical analysis to correlate these tweets.
This attack can be mitigated by introducing random delays between the moment the tweet is sent to the different fake accounts. However, depending on the length of the delay and the situation, this may be undesirable

Linking account by IP-address The most obvious attack is to link an account to a user by determining which IP-address is used to log into different accounts that use Pwitter. If the same IP-address is used for multiple accounts, this may indicate that these accounts are controlled by the same person. If the SNO can determine which fake accounts belong to the same person, he then may trivially reconstruct the social network by looking at the users that follow or are followed by each 1-to-1 account.
Fortunately, various solutions for anonymity on the IP-layer exist, the most well-known being onion routing. An onion router, such as TOR (The Onion Router), prevents the server from seeing the IP-address of the connecting host by routing the request through different encrypted links between nodes of the TOR-network. If a new circuit is used to log

into each account, the connection to API is made from a random exit node, which makes it infeasible to link users together based on the IP-address.

Linking requests by timing Even if the IP-address could successfully be masqueraded, the SNO may be able to link requests to their server based on the timestamps. The approach is similar to the one described above: if the application sequentially logs into n different accounts within a short period of time and/or if the intervals between logging out account i and logging into account $i + 1$ are more or less constant, accounts may be linked using statistical analysis.
This attack may be mitigated by implementing random delays between connections to the API. However, this may have an effect on the usability of the application, because the may user may notice these delays.

3.2.5 Limitations

One of the biggest disadvantages of this model is that a considerable amount of accounts has to be created. While it is technically possible to create more than one account, it is rather difficult because only one account per email-address is allowed. This implies that a new email-address for each has to be created as well. While this can be done automatically, Twitter seems to have deployed various systems that detect the automatic creation of accounts, probably because these same methods are frequently used for spam.

3.3 Broadcast model

3.3.1 Introduction

The second approach tries to hide the social network, by exploiting one (or a few) central, shared and public accounts. Instead of following different accounts, there is one account that is controlled by individual users. Similar to the accounts-model, this model provides an interface between the user and Twitter. Because of the public nature of the shared account, messages are encrypted in such a way that only authorised followers can decrypt them.

In this model we intend to use a central and shared Twitter account, called 'the broadcast account'. This account will be used to broadcast all tweets by participating users. In the situation where Alice wants to follow Bob, Alice does not need a direct connection with Bob anymore if she is following the broadcast account and Bob sends his tweets via the broadcast account.

Encryption is needed to protect Bob's tweets. Bob will need to be in control who is able to read his messages and who is not. Although it seems straightforward to encrypt his message with one of the well-known encryption methods, it is quite a challenge to encrypt in such a way that all communication takes place within Twitter.

We must realise that any form of direct communication between two parties, whether it is encrypted or not, already reveals the relation to the SNO, so no direct communication can take place. The problem is that encryption methods require a key establishment, which in turn rely heavily on direct communication between the participating parties. A detailed solution of methods and algorithms for this problem is given in chapter 5.

Channels

The public broadcast account is used as a broadcast channel for the messages by users. But as we will explain later on, more message types are needed as well. Besides messages holding the content of tweets, we also have messages containing keys and follow requests.

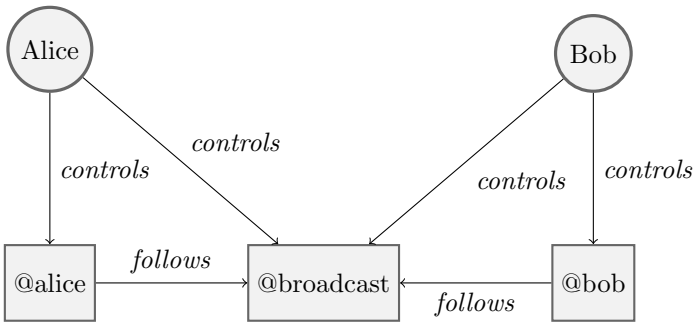
To be able to distinguish the different messages, multiple solutions exists to separate the messages:

- *Hash tag*: By applying a hash tag per message type, we can utilise Twitter API search capabilities to easily filter in the different messages in the public broadcast channel.
- *Multiple accounts*: A more robust solution would be to add a separate public account per message type. In this case every account stands for a separate channel.

3.3.2 Design

In figure 3.6, a schematic overview of the broadcast model is given. Instead of figure 3.1, we use two identities (Alice and Bob) to clarify this model. For communicating with Bob, Alice uses her normal account, which follows the broadcast account. All communication between Alice and Bob will be indirect and going through the broadcast account. Note that none of the accounts in the overview follow anyone besides the broadcast user.

Figure 3.6: Broadcast model schematic overview



3.3.3 Protocol description

The design of the model can be divided into two phases:

I Establishment of the following relation.

II The tweeting process.

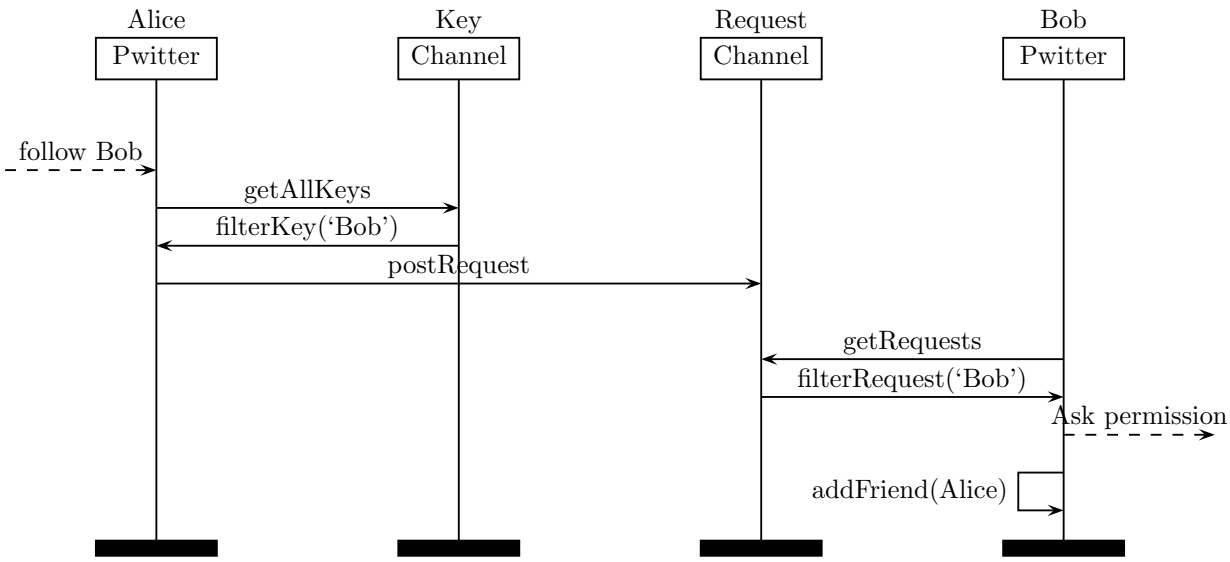
Each of these phases have their own particular characteristics, thus will be explained separately below.

I. Relation establishment

To establish a hidden follower-relationship between two parties in the broadcast model, the receiver R (the one who wants to read messages from somebody) needs to perform a request to the sender S to add him to his hidden friend list. As will be described in more detail in chapter 5, the protocol of the broadcast model will then ensure that R can read the encrypted messages in the shared account.

Figure 3.7 shows the relation establishment process. For this inprocess, we need an extra channel C_{req} containing all these follower requests. Receiver R will use the public key of the sender K^S_{public} to send out a request on the request channel C_{req} containing the encrypted name of the sender S he wants to follow. All Pwitter clients monitor this channel and try to decrypt the messages with their private key to see if a message contains their own name. If so, the request is shown in the GUI of the Pwitter. On acceptance, the sender of the request is added to the list of friends and new tweets will contain the secret key in the provided access control list.

Figure 3.7: Relationship establishment of broadcast model



II. Tweeting

Sending out a tweet is done by sending an special crafted encrypted message to the message channel C_{msg} of the broadcast account. The hidden list of friends, stored within Twitter but available in Pwitter, is needed to construct this message. The details of the algorithm to construct this message are described in detail in chapter 5.

Figure 3.8 shows the process of tweeting. In this model only one tweet is sent to the broadcast channel, containing an encrypted and encoded tweet consisting of both the list of designated

receivers and the original message. The encryption and encoding is done for all Alice's followers at once.

In the broadcast model, constructing the timeline starts by retrieving all new messages in the broadcast account. Because the broadcast account is used by all Pwitter-users, it will contain messages not designated for the retriever. Only the messages which can be succesfully decrypted will be presented on the Pwitter client, as shown in figure 3.9.

Figure 3.8: Tweeting of broadcast model

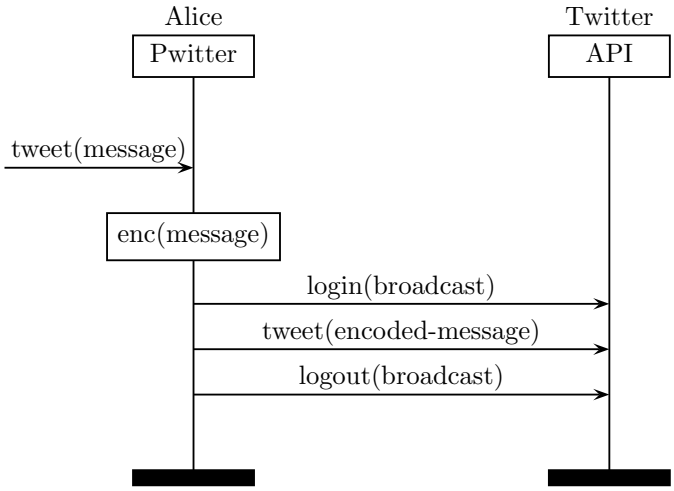
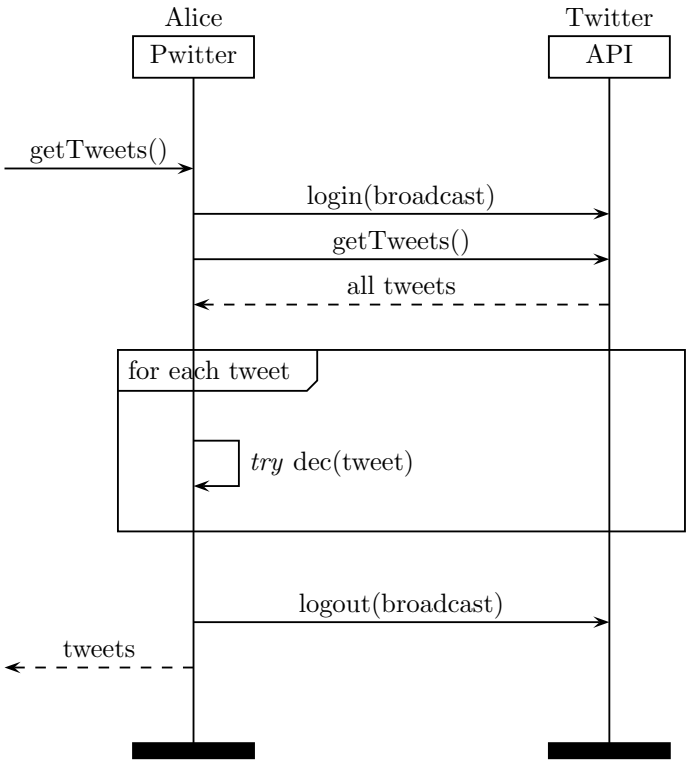


Figure 3.9: Constructing timeline of broadcast-model



3.3.4 Scope of the afforded privacy

Although the broadcast-model is not formally verified in this thesis, we will give a brief and informal analysis of the afforded privacy in this section. We will first analyse the privacy of the model in terms of the concepts that were introduced in section 2.1. Then, we will present a risk analysis, including mitigations for those risks.

Privacy of the broadcast-model

In chapter 5, we will explain that we have two versions of the broadcast-model. The first version provides both sender and receiver anonymity, but in the second version, the sender anonymity is given up for practical reasons. Thus, only receiver anonymity remains. This version is also the one that we used in our proof-of-concept and has some advantages in terms of efficiency and implementation feasibility. As long as the broadcast-model protocols prevents linkable actions, receiver anonymity is guaranteed.

For the relation establishment we achieve unlinkability by requesting all keys and requests, and perform filtering on the client outside the view of the SNO. This way, the SNO can not determine which keys and requests are of interest of the requestor. The same idea can be found in the timeline construction protocol. All tweets are downloaded to prevent linkability. For

tweeting, no recipient is directly involved in the process.

Risk analysis

Attack on stored data The SNO could try to perform a direct attack on the storage of the list of friends itself.
As described in section 3.1.1, we store the list of friends within Twitter itself, and preferably within an image. The security of the list depends mainly on the chosen encryption methods and strength of these methods. Besides that, the user needs to choose a password for this encryption as seed or salt and the security depends on the ability to choose a good password as well.

Traffic analysis Instead of this direct approach, the SNO could try to retrieve the social network by reconstructing it from traffic on different layers of the OSI layer. Especially when looking at the provided API, the SNO could analyse the used search calls when we utilise their API. Without carefully designed algorithms, we could easily reveal relationships. On the network layer, IP address and timing related issues, can reveal connections in some of the implementations of the broadcast model.
The counter measures for these kind of attacks are all addressed in detail in chapter 5.

3.3.5 Limitations

Because everybody uses the same account to communicate, congestion is likely to happen when the model is used by too many people. This is due to the fact that Twitter has API usage limitations, especially on the number of API requests per minute and the number of messages which can be returned per request. When n Pwitter users submit m messages per minute, their will be $n * m$ new messages in the broadcast account per minute. This will lead to problems when speed of the new messages exceeds the API speed.

To overcome these problems, a smart clustering of multiple number of shared broadcast accounts could help.

Chapter 4

Formal verification

4.1 Introduction

The models developed in this thesis aims to hide the user's social network from the SNO by implementing a privacy-enhancing layer. This means that the tool is only useful if the user can be sure that the desired information is hidden from the SNO. While encryption may play a vital role in realising this goal, it is still only a part of the system. Correctly designing this system is just as important as using the proper cryptographic algorithms. The analogy that Cremers presents in his dissertation is clarifying [Cre06]:

> If you have the perfect bike lock and chain, but fix the chain around the bike in the wrong way, somebody can still steal your bike. In much the same way, the security of a computer system depends on the way the components interact. Encryption is like a bike chain. It is a useful mechanism that can be used in the construction of secure systems, but you can apply it in the wrong way. It is not a guarantee for security by itself.

While the cryptographic algorithms used in this proof-of-concept are industry standards, the protocol is uniquely designed. Unlike its building blocks, it is therefore not thoroughly tested for security. To have some assurance about the correctness of the protocol, it can be modelled by a formal modelling language and subsequently verified. In this work, we will use the applied pi calculus to do this. We should note that the scope of this formalisation only encompasses the application layer. While it might well be possible for an adversary to obtain information about the social network using different layers (for example, the IP-address that was used to log into different accounts), there are methods to remain anonymous on this level. The IP-address can be hidden using TOR for example. Onion routing has also been formally verified by [MVdV04].

Formal modelling has been used for security protocols for quite some time. The applied pi calculus has been developed to verify security in particular, and has been described in detail in [RS11, AF01]. Also the verification of privacy-related concepts has been extensively studied. For example, the privacy of voting protocols has been verified in [DKR09, KR05] and also other properties such as coercion-resistance and receipt-freeness have been verified in relation to voting by [DKR06] and also in a more general context by [DJP13]. While these papers contain useful notions for the modelling of privacy of the social network, we also rely on work done in [ACRR10], where anonymity and unlinkability is modelled in the applied pi calculus. Lastly, some relevant observations about the hiding of identities in security protocols have been made in [FA03].

4.1.1 Contribution

In this chapter, we define a framework for OSNs, which clarifies how an SNO could obtain information about the social network in the application layer and adopt a terminology to facilitate reasoning about the privacy aspects of the social network. This should be a foundation that can be used for modelling social networking in formal modelling languages. Furthermore, we model the functioning of Twitter in the applied pi calculus, which allows privacy-enhancing layers upon Twitter to be formally verified. Lastly, one of the privacy-enhancing layers that is developed in this thesis will be tested for a possible way to obtain information about the social network using Blanchet's ProVerif tool [BAF08, Bla09].

4.2 Concepts

In this section, we will define various relevant concepts more formally. This allows us to model these concepts correctly in the applied pi calculus. We will start with giving a formal definition of the social network, after which we will look at how Twitter can obtain information about the social network. Finally, we will introduce various concepts that help to define the privacy of the social network.

4.2.1 Social network

In social network analysis, a social network is often modelled as a graph. Besides the fact that graph theory provides an intuitive way to model a social network, its formal definition is based on set theory, which is useful to model social relations in applied pi. In figure 4.1, we see an example of a social network represented as a simple graph. Each node represents a person and each represents a social relation. Thus, an edge from Alice to Bob means that Alice and Bob know each other. Formally, we define a social network as follows:

Definition 1 (Social network)**.** *A social network graph is an ordered pair $G = (V, E)$, comprising the set V of nodes, which represent persons and the set E of edges, which are 2-element subsets of V. An edge of the form $\{x, y\}$ represents a social relation between x and y.*

Applying definition 1 to the social network that is depicted in figure 4.1, we have a set G, which consists of the set of persons $V = \{Alice, Bob, Charlie, Dave, Frank\}$ and the set of edges $E = \{\{Alice, Bob\}, \{Alice, Charlie\}, \{Bob, Charlie\}, \{Dave, Bob\}, \{Dave, Frank\}, \{Frank, Alice\}\}$. Each element in E represents a social relation.

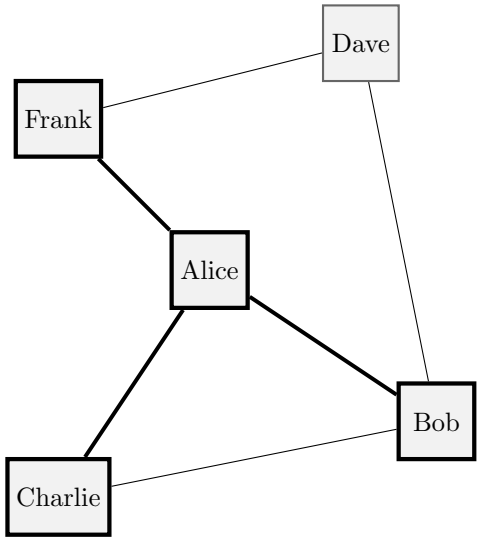
We can further specify the definition of a social network to the social network of a specific person:

Definition 2 (Person's social network)**.** *If a person A is part of a social network G, thus $A \in V$, this person's social network is a social network as well and is denoted $G_A = (V_A, E_A)$, where $V_A \subseteq V \wedge E_A \subseteq E \wedge \forall x \in E_A, A \in x$.*

In figure 4.1, the social network of Alice is accentuated. Applying definition 2, we can see that the social network of Alice is the set G_{Alice}, which consists of the set of persons $V_{Alice} = \{Alice, Bob, Charlie, Frank\}$ and the set of edges $E = \{\{Alice, Bob\}, \{Alice, Charlie\}, \{Frank, Alice\}\}$. It should be pointed that this definition of person's social network only entails *direct* connections. Thus, a 'friend of a friend', is not considered part of a person's social network. While it can be argued that this should be included in the definition of a social network, we do not aim to give a definition that is generally applicable, but one that suits are needs for modelling a social network in the applied pi calculus.

The definition of a person's social network implies that if we consider the social network G_x of some person x, a person y is in his social network if there exists a social relation between x and y.

Figure 4.1: A social network



4.2.2 Social relations in Twitter

When applying social network analysis on Twitter, we can depict the 'following-relations' as a graph as well. In figure 4.2, an example graph of a social network on Twitter is given. The graph is similar to the graph of a social network that was shown in figure 4.1, the difference being that this a directed graph. This means that the definitions of a social network on Twitter are slightly different from earlier provided definitions of a social network and a person's social network:

Definition 3 (Social Twitter network)**.** *A social Twitter network is an ordered pair $G = (V, E)$, comprising the set V of nodes, which represent accounts and the set E of edges, which consists of ordered pairs of nodes. An edge of the form (x, y), where $x, y \in V$, means that account x follows account y.*

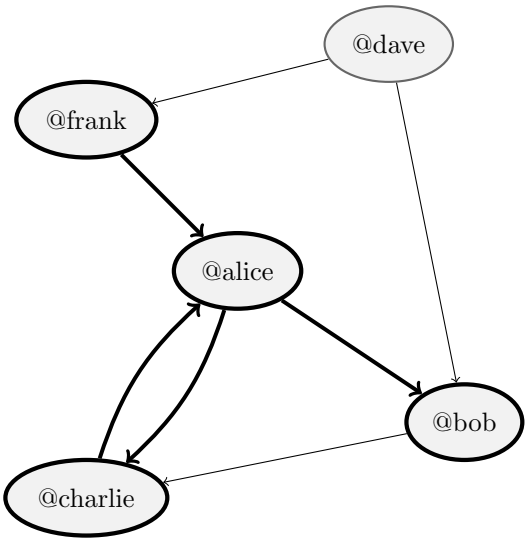
Definition 4 (Person's social Twitter network)**.** *If an account A is part of a social Twitter network G, thus $A \in V$, this account's social Twitter network is a social Twitter network as well and is denoted $G_A = (V_A, E_A)$, where $V_A \subseteq V \wedge E_A \subseteq E \wedge \forall (x, y) \in E_A, (x = A \vee y = A)$.*

It might be worth pointing out that the indegree of a node represents the number of followers of that account, and the outdegree represents the number of people that this account is following.

Having defined a social network that consists of persons and a social network on Twitter that consists of accounts, we should explain how these are related. Each account is referenced with the username on Twitter. The username that a person chooses is a pseudonym, or more precisely: a person-pseudonym. This is defined as:

Definition 5 (Person-pseudonym [PH10])**.** *A person-pseudonym is a substitute for the holder's name which is regarded as representation for the holder's civil identity.*

Figure 4.2: Social network in Twitter



It's worth noting that a person-pseudonym in itself can provide a certain degree of anonymity. However, person-pseudonymity is considered to provide a weak notion of anonymity [PH10]. This makes sense, because the identity can quite easily be revealed in different ways, for example by linking the person-pseudonym to an identity that is using the social network, as has been demonstrated by [NS09]. Therefore, we will assume that an adversary can always find out the identity of a person-pseudonym[1]. This allows us to consider a 'following-relation' between two person-pseudonyms in Twitter as a proxy for a social relation as defined in definition 1. More formally: if a person-pseudonym A follows another person-pseudonym B, this indicates that a social relation as defined in definition 1 between the identities that control A and B exists.

While the 'following-relation' of person-pseudonyms in Twitter is the most obvious indicator that a social relation exists between the controlling identities, there are other ways in which a social relation between identities can be discovered. For example, if @Alice sends a direct message to @Bob, this may give away that @Alice and @Bob know each other. To define this more formally, we introduce the definition of *linkability*:

Definition 6 (Linkability [PH10])**.** *Linkability of two or more items of interest (IOIs, e.g., subjects, messages, actions, ...) from an attacker's perspective means that within the system (comprising these and possibly other items), the attacker can sufficiently distinguish whether these IOIs are related or not.*

For Twitter, we consider the following IOIs to have the potential to give away a relation between two exemplary users @alice and @bob:

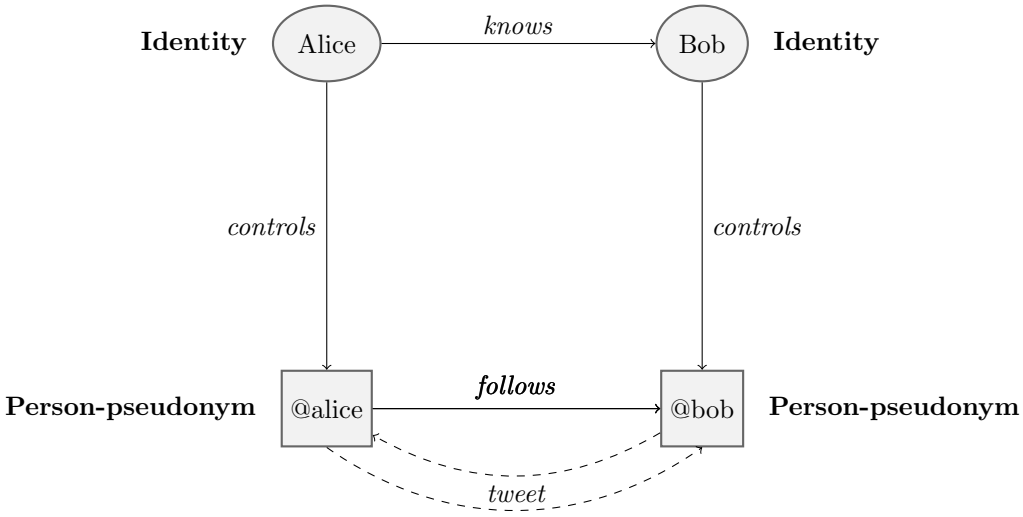
- Tweets
- Likes
- Retweets

[1]Note that this does not imply that all usernames are person-pseudonyms, as there exist other pseudonyms as well, for example relationship-pseudonyms (see section 4.3).

- Direct messages
- Replies

To illustrate the relation between a social relation and a 'following'-relation on Twitter, figure 4.3 is provided. In this schema, Alice and Bob represent two exemplary persons that use Twitter. Alice and Bob both have regular Twitter account, respectively '@alice' and '@bob'. These accounts are considered person-pseudonyms. Between these two accounts, tweets are sent. These can be linked to the sender, the recipient or both.

Figure 4.3: Privacy framework



4.2.3 Privacy of social network

Having defined the social network and how information on Twitter may indicate the existence of social relations, we should define what is meant by *privacy of the social network*. Since privacy is often used as an umbrella concept for privacy-related concepts, such as anonymity, unlinkability and unobservability, it can be unclear what is meant exactly by privacy. We will define privacy of the social network as the extent to which an adversary can obtain information about a person's social network. This implies that we have privacy of the social network if this information cannot be obtained. We will specify this by separately considering two different possible ways in which an adversary can obtain information about the social network: the 'following'-relations and the linking of IOIs.

Following-relations

As pointed out in the previous section, we assume that a 'following-relation' of two person-pseudonyms indicates the existence of a social relation. This implies that if the SNO knows that x follows y, where x and y are both person-pseudonyms, he knows that there exists a social relation between x and y. Thus, we consider a 'following-relation' of the form (x, y) as something that the SNO should not know: a secret. Secrecy can be modelled in applied pi as a reachability

problem [RS11]. We will further elaborate the modelling of this problem in the applied pi calculus in section 4.5.

Linking items of interest

While the 'following-relation' is defined relatively straightforward as a secrecy problem, the linking of items of interest is slightly more challenging. To do this, we should first give a definition of unlinkability, which is the negation of definition 6:

Definition 7 (Unlinkability [PH10])**.** *Unlinkability of two or more items of interest (IOIs, e.g., subjects, messages, actions, ...) from an attacker's perspective means that within the system (comprising these and possibly other items), the attacker cannot sufficiently distinguish whether these IOIs are related or not.*

In section 4.2.2, we already provided a list of IOIs that are considered relevant for obtaining information about the social network. Of these IOIs, the sender and recipient are particularly relevant since we are aiming to detect whether these can be linked. We can define the link between the message and sender or receiver as respectively *sender anonymity* and *recipient anonymity*:

Definition 8 (Sender anonymity [PH10])**.** *Sender anonymity of a subject means that to this potentially sending subject, each message is unlinkable.*

Definition 9 (Recipient anonymity [PH10])**.** *Recipient anonymity of a subject means that to this potentially receiving subject, each message is unlinkable.*

Since our aim is obtain privacy of the social network, which consists of social relations, we need to obtain *relationship anonymity* between Alice and each of her relations. Relationship anonymity is defined as:

Definition 10 (Relationship anonymity [PH10])**.** *Relationship anonymity of a pair of subjects, the potentially sending subject and the potentially receiving subject, means that to this potentially communicating pair of subjects, each message is unlinkable.*

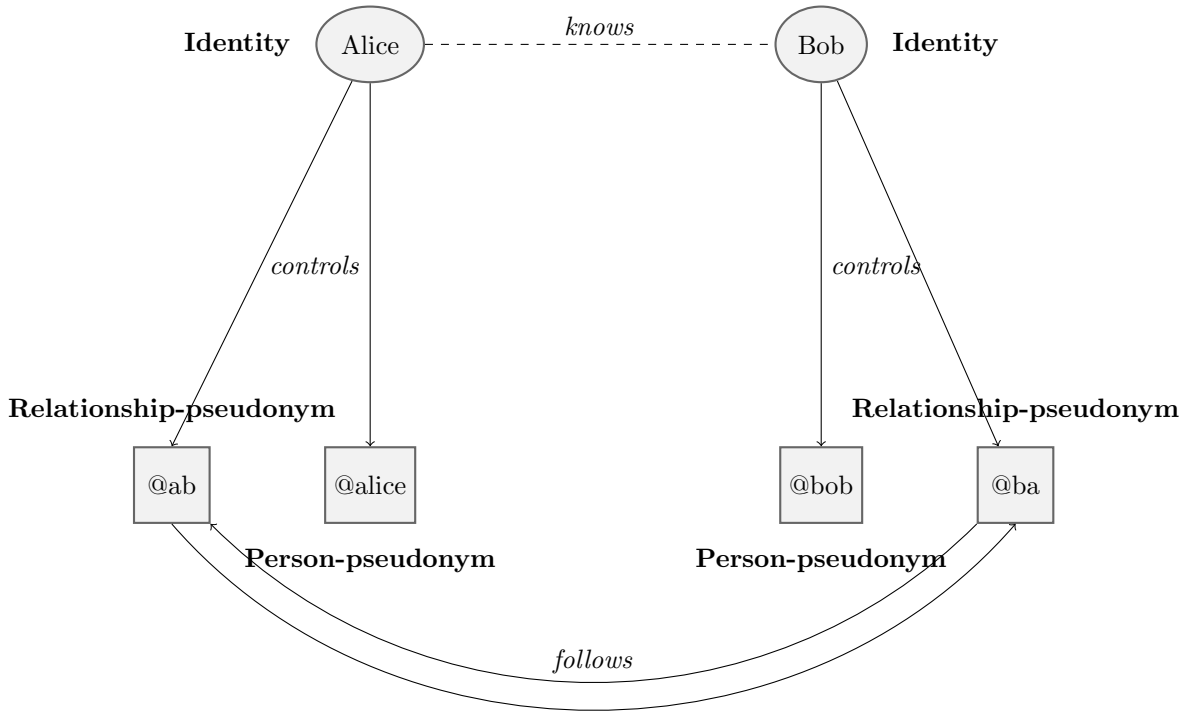
As Pfitzman & Hansen point out, relationship anonymity is weaker than each of sender and recipient anonymity: if we have either sender or recipient anonymity, this implies relationship anonymity. While the authors state that it does not matter if these alternate, we should keep in mind that this does not hold when the messages can be linked to each other. If we can prove that at least one of these always holds, relationship anonymity is proved. If both fail it is still possible that relationship anonymity exists, namely if the same message never can be linked to both sender and receiver at the same time, and the message cannot be linked to other messages that are sent between the same sender and recipient. Summarising, the following scenarios are possible:

1. Sender anonymity: messages can never be linked to sender. This directly implies relationship anonymity.
2. Recipient anonymity: messages can never be linked to receiver. This directly implies relationship anonymity.
3. Mixed anonymity: this implies relationship anonymity if the following conditions are met:
   (a) messages cannot be linked to sender and receiver at the same time.
   (b) messages that can be linked to sender cannot be linked to messages that can be linked to the receiver.
   (c) messages that can be linked to receiver cannot be linked to messages that can be linked to the sender

4.3 Protocol

As mentioned in the introduction, we will only conduct a formal verification of the tweeting process of the accounts-model. A detailed description of this protocol has been provided in section 3.2.3. To be able to distinguish between person-pseudonyms and the 1-to-1 accounts that are used in the account model, we will consider the 1-to-1 accounts *relationship-pseudonyms*. This means that a different pseudonym is used for each relationship. In figure 4.4, the model that was presented in figure 4.3 is extended with the notion of relationship-pseudonyms.

Figure 4.4: Privacy framework



4.4 Applied pi calculus

The applied pi calculus [AF01] is language that can be used to describe and verify processes and their interactions. It is based on the pi calculus and has similarities with the spi calculus [AG99]. The main difference with the latter is that the spi calculus has a fixed set of primitives, while applied pi is defined by an equational theory. This provides much more flexibility to model non-standard protocols. It is especially suitable for the verification of security protocols and can be used for the verification of privacy as well [KR05]. The verification process can be done manually or automatically, using the ProVerif tool [BAF08, Bla09]. As pointed out by [KR05] and [Bla04], especially the possibility to automatically prove that certain processes are observationally equivalent makes ProVerif useful for proving privacy. An example of the

automatic verification of privacy has been conducted by [DRS08].

The basic syntax and semantics of the applied pi calculus are described in appendix B. For a more detailed explanation of its characteristics, see [AF01] and [RS11]. Furthermore, we will consider the SNO, Twitter in this case, to be a standard Dolev-Yao adversary [DY83]: it can eavesdrop all communication and actively block, create, alter and inject messages. As is standard in the applied pi calculus, we will assume that all channels are anonymous.

4.5 Modelling in applied pi calculus

4.5.1 Twitter context

Before formalising the privacy aspects, we will define a context in which a normal or private tweeting process operates. In this context, the interaction with the user and Twitter is modelled. We have plain process T. The repetition of the role represents multiple users. The Twitter-process T is built up from sub-processes, which will first be explained.

- *User process* U. The process U represents the user-process, which models the interaction with the user. This process consumes and produces the tweets. We use a control interface between the user process and the tweeting process, which is denoted $\mathcal{V}$ and uses the channels *login*, *send*, *req* and *read* to respectively log the user in by sending the 'following'-relations, send tweets, request new tweets and receive the tweets. We define $\mathcal{V}_X = \{send_X, read_X, login_X, req_X\}$. Thus, when a user logs in, he first transmits his relations on channel *login*. Then, if a person types a tweet, it is sent by the process U on channel *send*. Similarly, if the users requests his timeline, he sends a request on channel *req*. The response will be received on channel *read*. Since these channels are private they will be placed under restriction.

- *Backend process* B. The backend of Twitter is defined by the process B. In this process, incoming tweets are sent to the users that follow the sender of the tweet. This process listens on the public channel f for following relations, on the public channel c for tweets and on the public channel r for requests. If the process B finds a relation, tweet and request, where the subject of the following relation matches the requestor and the sender of the tweet matches the 'follows'-part of the relation, the tweet is forwarded to the channel t

- *Interface context* $I[\cdot]$. Since the process B does not have access to the channels of $\mathcal{V}$, it cannot directly communicate directly with the user process U. To connect the user to the backend, an interface is needed. Since the interface is exactly what we will redefine in our privacy-enchancing protocol, this is modelled as a context $I[\cdot]$. The hole can be filled with a process that defines that standard functionality of Twitter, but also with a process that models our privacy protocol.

Before moving to the definitions in the applied pi calculus, we should define the signature and equational theory. For the standard Twitter functionality, we will need *tweets* and *relations*. The first represents a tweet of the form (i, m), where i is the sender and m the message. The second is a 'following'-relation of the form (i, i'), where i is the subject and i' as the user he follows. This results in the signature $\Sigma = \{$tweet, sender, message, subject, relation, follows$\}$, which is

equipped with the equational theory E:

$$\begin{aligned} \text{sender}(\text{tweet}(i, m)) &= i \\ \text{message}(\text{tweet}(i, m)) &= m \\ \text{subject}(\text{relation}(i, i')) &= i \\ \text{follows}(\text{relation}(i, i')) &= i' \end{aligned}$$

Note that requests are not defined in the equational theory, as these simply consist of the username that makes the request.

The Tweeting-process is then defined as follows:

$$\begin{aligned} T &\triangleq !\nu i.\nu i'.\nu \mathcal{V}.(!U \mid !I[\cdot] \mid !B) \\ U &\triangleq \overline{login}\langle \text{relation}(i, i')\rangle \mid \nu m.\overline{send}\langle \text{tweet}(i, m)\rangle \mid \overline{req}\langle i\rangle \mid read(w) \\ B &\triangleq f(x).r(y).c(z).\text{if subject}(x) = y \text{ then if sender}(z) = \text{follows(x) then } \bar{t}\langle z\rangle \end{aligned}$$

The context $I[\cdot]$ can be filled with a process that represents the private tweeting process. When using normal Twitter, this process is built up from a few simple processes that forward information from the private channels in $\mathcal{V}$ to the public channels that are used by the Twitter backend:

$$I[login(x').\overline{f}\langle x'\rangle \mid tweet(z').\bar{c}\langle z'\rangle \mid req(y').\bar{r}\langle y'\rangle \mid t(w').\overline{read}\langle w'\rangle]$$

When a user logs in, he transmits his 'following'-relations on the private channel *login*. Whenever input is received on this channel, it is immediately transmitted on the public channel f. Similarly, when a user sends a tweet on the channel *send*, it is transmitted on the channel c, which is a public channel where all tweets are sent to. If a request to fetch tweets for the timeline is received on channel *req*, it is forwarded to the channel r. If the process B finds a relation, tweet and request, where the subject of the following relation matches the requestor and the sender of the tweet matches the 'follows'-part of the relation, the tweet is forwarded to the channel t. All tweets on channel t are simply forwarded to the channel *read* in the normal tweeting process.

Our model is not entirely accurate since each time a user logs in, information about the relations is submitted. Normally, this is done only once, as Twitter saves this information. We slightly deviate from this reality because we do not want to complicate the model unnecessarily with an extra registration-process. Since a standard Dolev-Yao attacker is assumed to never forget, this does not have consequences for the verification.

4.5.2 Private tweeting

As mentioned in section 4.1, there exist two possible approaches that the adversary can take to learn something about the user's social network. We will only verify one of these in the applied pi calculus, namely by obtaining information about the 'following'-relations. This information is sent to Twitter during the login process. When using Twitter, it is rather easy to prevent Twitter from discovering information about the social network this way, namely not following

someone. The problem is that Twitter becomes unusable, especially if protected tweets are used (which we generally assume in this thesis). This implies that if we can model a functioning communication process that allows the user to interact the same way with Twitter (using $\mathcal{V}_X$), while no following relation for the person-pseudonym exists, we can prove that relationship anonymity is not compromised this way.

For simplicity and readability, our model only works for one following relation. However, since we have defined the social network as a graph that is made up from social relations, proving that one following relation remains hidden, proves that social network can be hidden.

When testing for reachability-based secrecy in the applied pi calculus, we normally define an intruder process and check whether this process is capable of sending the secret on a public channel. Since the adversary in our model is Twitter, we will consider the backend-process B as the intruder-process. When using Twitter, the process B obtains the 'follower'-information from public channel f. This means that the process B already does what we would expect from an intruder-process. We will simply have to check whether the 'following'-relation we are interested in can be output on a public channel.

Definition 11 (Privacy of a following-relation)**.** *A Twitter protocol satisfies privacy of a following-relation M if there is no plain process B such that* $(\mathrm{fn}(B) \cup \mathrm{bn}(B)) \cap \mathrm{bn}(P) = \emptyset$ *and* $P \mid B$ *can output* M*.*

4.5.3 Model

In this section, we will define the process that represents the protocol of the accounts-model that is developed for this thesis. This process will fill the hole of the interface-context $I[\cdot]$, which should result in a process where tweets can be sent to other users. The privacy properties will be verified in section 4.6.

First of all, we will have to extend the signature and equational theory with an encryption and decryption function. These are standard symmetric encryption and decryption functions, resulting in the signature $\Sigma = \{\text{tweet, sender, message, subject, relation, follows, enc, dec}\}$, which is equipped with the equational theory E:

$$\begin{aligned} \mathrm{sender}(\mathrm{tweet}(i, m)) &= i \\ \mathrm{message}(\mathrm{tweet}(i, m)) &= m \\ \mathrm{subject}(\mathrm{relation}(i, i')) &= i \\ \mathrm{follows}(\mathrm{relation}(i, i')) &= i' \\ \mathrm{dec}(k, \mathrm{enc}(k, m)) &= m \end{aligned}$$

Since we are only modelling the tweeting process, we assume that two users who are privately tweeting have a shared key k. We define the process P, which is the Pwitter process. This process will fill the context $I[\cdot]$ and consists of the following sub-processes:

- *Login process L.* This process receives input of type relation(i, i') on channel *login*. It then outputs the 'following'-relations of type relation$(fake_i, fake_{i'})$, where $fake_i$ is the user-name of the relationship-pseudonym that this user has to communicate with the relation-pseudonym that i' uses: $fake_{i'}$.

- *Tweeting process S.* This process handles tweets that are sent by the user that is currently logged in. It waits for input on the private channel *send*. When it receives input, it encrypts the tweet and encapsulates it in a new tweet, where the sender is $fake_i$. Finally, it outputs this tweet on channel c.
- *Reading process R.* This process handles incoming tweets. It listens for requests on channel *req*. If a request from user i is received, it sends a request for user $fake_i$ to the channel r. Furthermore, it listens on channel t and tries to decrypt any incoming tweets with the key that belongs to this relation-pseudonym. The decrypted tweet is then sent on the private channel *read*.

$$
\begin{aligned}
P &\triangleq \nu fake_i.\nu fake_{i'}.\nu k.(L \mid S \mid R) \\
L &\triangleq login(x').\bar{f}\langle \text{relation}(fake_i, fake_{i'})\rangle \\
S &\triangleq send(z').\bar{c}\langle \text{tweet}(fake_i, \text{enc}(k, z'))\rangle \\
R &\triangleq req(y').\bar{r}\langle fake_i\rangle \mid t(w').\overline{read}\langle \text{dec}(k, w')\rangle
\end{aligned}
$$

4.6 Analysis

The processes have been rewritten as ProVerif scripts. While these may slightly deviate from the models in the previous section, they are semantically equivalent. The scripts can be found in appendix C. For both the normal tweeting process and the private tweeting process, we first verified whether the tweet was received using the *event* syntax. Since ProVerif also checks whether an attacker could insert a fake message, this check was conducted while all channels were modelled private, which forced ProVerif to find a trace where a tweet was sent from a user i, which could be read by a user i' if i' follows i for both the normal tweeting process and the private tweeting process.

Secondly, reachability of the secret create_relation($alice, bob$) was tested, where *alice* and *bob* are considered person-pseudonyms. As expected, this test failed for the normal tweeting process and passed for the tweeting process. This proves that our model satisfies privacy of the 'following'-relation.

4.7 Conclusion

In this chapter, we aimed to formally verify one of the privacy-enhancing layers that was developed in this thesis. In order to do so, many of the concepts that were coined earlier, such as *social network, social relation* and *relationship anonymity*, needed to be defined more formally. Defining the social network as a graph allowed us to reason about the corresponding set, which makes it easier to model these concepts in the applied pi calculus. Furthermore, various possibilities to obtain information about the social network have been discussed. Perhaps the most obvious way to do so, is obtaining information about who a user follows. We have modelled the Twitter context and the accounts-model in the applied pi calculus in order to verify whether Twitter could access this information when the accounts-model is used. The latter was defined as a reachability-based secrecy problem. We have verified this using the ProVerif-tool and can conclude that the tweeting process of the accounts-model indeed satisfies privacy of the 'following'-relation.

While we hope to have made a contribution to the ongoing research on the formal verification of privacy-enhancing protocols, we realise that this work perhaps raises more questions than it answers. First of all, we have only modelled one way to obtain information about the social network in the applied pi calculus, while we pointed out in section 4.2 that there are multiple possibilities to obtain this information, such as linking messages. Further work should be conducted to verify these other possible attacks as well. Secondly, we have only verified the tweeting process, which is arguably the easier part of the accounts-model. The more complex relation establishment part of the protocol deserves to be formally verified as well, since the complexity indeed makes it more likely that it contains weaknesses that are overlooked when it is only informally analysed. Lastly, the models in this chapter have often been written to be correct for Twitter. This could be further improved by redesigning them in such a way that they are applicable to OSNs in general, so privacy-enhancing layers for other OSNs could be formally verified as well.

Chapter 5

Cryptographic solutions for the broadcast model

5.1 Introduction

In the search for a method to hide friends from a SNO, we found a way to perform this with a model we introduced as broadcast-model.

As described earlier, hiding the list of friends itself, with a privacy enhancing layer, is achievable and we could even store the network within Twitter itself. Of course, just hiding the network, a list of contacts, is not the main goal. Some interaction with the hidden network are required to make the solution usable: reading the messages from the hidden friends and send out tweets to your hidden friends. As shown in section 3.3 this ultimately requires a form of encryption to the messages that are published in the broadcast account.

The need for encryption in the broadcast channel, in combination with our main goal of hiding relationships, result in specific requirements of the encryption method. The properties of the encryption method should be:

- Group-based: It is very inefficient to send one encrypted message to each receiver. Therefor we need an encryption method that supports group-based encryption that allows the sender to send one message to a group.
- Dynamic membership: Because the list of friends are likely to change once in a while, the set of receivers is not fixed.
- Anonymous members: To prevent a receiver from sharing the social network (and thereby revealing the list of friends), receivers should not know each other.
- Offline: Receivers do not need to be online when a sender broadcasts a message. Group membership, key exchange or other communication between sender and receiver should be handled offline/asynchronously.
- No direct communication: Sender and receiver are not able to communicate directly within the environment. This could reveal a link between them. Therefor we need an encryption method that does not require direct communication between parties.

- No 3rd party: Besides the sender and the receiver, the encryption method should not rely (or need) a 3rd party for services like: storage and retrieval of keys or other data and communications. This requirement is due to the fact that we do not want to trust other parties.
- No out of bound: No communication between sender or receiver should be needed outside the normal usage of Twitter and Pwitter.

These set of requirements are challenging for current cryptographic algorithms. A detailed comparison between these methods and the requirements is made in section 5.3.

We will give a short re-introduction of the broadcast model and will first show the need for cryptography in this model and design considerations when using Twitter's API, continued by evaluating existing cryptographic solutions with respect to our requirements, and finally present our method. The final proof-of-concept of the broadcast model, will use cryptographic algorithms. Although the algorithms themselves are well known in literature, the protocol and methodology of applying it to this use-case is uniquely designed.

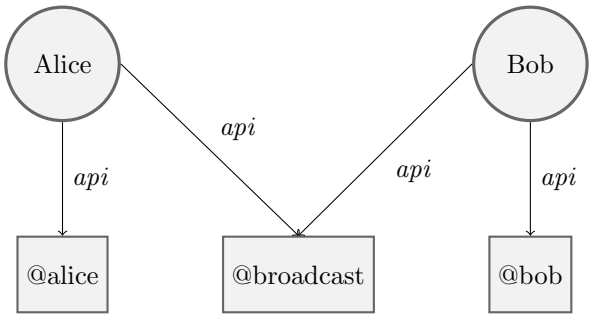
5.1.1 Contribution

We present a comprehensive set of concepts and algorithms to successfully hide relationships from a SNO while maintaining most functionality and by reusing the infrastructure already in place by the SNO. This is done with the use of a single shared account, called the broadcast account. Although we describe the process for Twitter, our method can be applied to other OSN as well.

5.2 Context

In the broadcast model and the privacy enhancing layer, we can find several key elements where the SNO could detect linkabilty, as described by Pfitzmann [PH10].

Figure 5.1: Broadcast model API overview



One of the challenges with unlinkability is the use of Twitter's API by the enhancing layer. Whatever model you assume (accounts-model or broadcast-model), the client will perform actions utilizing the API with credentials from the original user, fake user, or the shared (broadcast) user. This API usage is needed for reconstructing the original timeline of the user, or getting

messages from the fake user, or posting to the broadcast channel, etc. We must assume that the SNO is seeing all these API calls.

When trying to hide the social network, we must use the API carefully. The type of API calls may reveal more information to the SNO than intended. Especially API calls requesting specific user profiles, messages, etc. will introduce information of the social network to the SNO.

For example: Getting someone's specific user profile has to be done via the API and these API calls are always performed in the context of a API user.

Our Pwitter client will need some API calls within the context of the user account as well as the shared account. The SNO could perform some simple IP address matching in combination with timing to link Broadcast channel actions to a specific user.

To prevent the SNO from reconstructing the social network based on linkable actions, Pfitzmann [PH10] showed that only sender *or* receiver anonymity is required. That's why there are several approaches to continue. One approach is to guarantee receiver anonymity only (and let the sender be public), which relaxes the use of the API because we do not need to hide the senders activity's on the broadcast account. In the following sections, we will use this approach. Another option could be to try to make both receiver and sender anonymous. In section 5.5 we give some key ideas achieving that with the broadcast model.

The use of the API can be seen as a set of read/write operations, in other words, it can be considered as I/O. Looking at the *read* operation (like getting messages via the API) we can identify two kinds of read operations:

- Type I: read operation for specific users, items, data.
- Type II: read operation for bulk transfer (read all, read all from date x, etc).

To prevent linkability on the API level, we will only use *read* operations of type II. This allows us to give up sender anonymity whilst retaining unlinkability. For *write* operations, it is sufficient to exclude receivers in any form of readable content, to prevent unlinkability. Messages in the broadcast account should be encrypted in such a way that only the designated receivers are able to decrypt it.

Summary Our design of the proof-of-concept of the broadcast-model assumes that we use receiver anonymity only, revealing the sender. We must assure that only use read operation of the category II are used and write operations must be encrypted. This combination guarantees that we reveal only one party and that the connection between sender and receiver is unlinkable.

5.3 Existing cryptograhic primitives

As shown in the introduction, we need an encryption method that ensures that our messages in the public broadcast channel are only read by the designated receivers. We will look for the simplest solution that performs this encryption which also conforms to our list of requirements, which are: group-based, dynamic membership, anonymous membership, offline, no direct communication and no 3rd party/central infrastructure. Finding the simplest solutions means that we look for a cryptographic primitive which complies to the requirements, and otherwise, look for the simplest combination of cryptographic primitives.

5.3.1 Cryptograhic primitives

Some of the well-known cryptograhic primitives are:

- Symmetric key cryptography: which uses the same key to encrypt and decrypt. An example is AES.
- Public key cryptography: also known as asymmetric cryptography, which uses a different key for encryption and decryption. An example is RSA.

Both symmetric key an public key cryptographic primitives lacks the built-in support for group-based encryption with dynamic and anonymous membership. They also do not describe how to distribute the keys between sender and receivers.

5.3.2 Broadcast encryption

In 1994, Moni Naor and Amos Fiat [FN01] introduced the concept of broadcast encryption. This form is widely used for protecting media like TV programs or DVD where the problem is to deliver protected content to a changing group of (paying) receivers. Instead of directly encrypt the content for the right specific users, the broadcast encryption schemes send special key information which helps the designated users to reconstruct the used encryption key, while leaving others in the dark. This initial concept was later enhanced by Noar in 2001 [NNL01] and Halevy in 2002 [HS02], reducing the needed storage, processing time and message sizes for this form of encryption.

Although the broadcast encryption scheme seems appropriate at first, and further communication can be done on broadcast channel, it requires an initial distribution of keys to the clients and management of these keys. The focus is more on revoking user from an initial set, then handling dynamic group membership. This would also require solutions like: direct sender/receiver communication for initial key distribution or OOB transfer of initial key's. Properties which do not fit with our requirements.

5.3.3 Attribute based encryption

In 2006, Goyal [GPSW06] described a method to encrypt data in a more fine-controlled way, based on shared attributes between sender and receiver. In his paper, he also describes a method for a broadcast-like system. The need of pre-labelling does not fit with our requirement of dynamic membership. Also the key distribution requires a no direct communication between sender and receiver. This make this cryptograhic solution not suitable for the broadcast model.

5.3.4 Multicast encryption

Canetti [CGI+99] presents some efficient solutions on multicast encryption, a group encryption method where membership can change efficiently. The paper and method focus on IP multicasting in the network layer of the OSI model. The setup requires a central management and the ability to live communicate with party's. Due to the offline requirement, multicast encryption is not applicable for the broadcast model.

5.3.5 Summary

Although multiple encryption schemes exists, a single cryptographic primitive that fit our use case is not available. In the following section we will present a method which combines symmetric key and public key cryptography that respects all our requirements.

5.4 Our method

Our method starts by encrypting the message with a symmetric key. For each receiver, this symmetric key is encrypted with a public key algorithm, using the public key of the receiver. The list of receiver keys is send together with the encrypted message. Both ciphertexts (message and list) are encoded in images and send to the broadcast channel with the sender's name. Each receiver knows to look for messages from this sender, and will only attempt decryption of messages intended for him. The receiver will find the used symmetric key in the provided list of keys, by using his private key, and use the symmetric key to decrypt the ciphertext.

Before we give more details about our method for applying encryption in this broadcast model, we need to introduce some concepts (Channels). Besides that we will give a complete overview of the setup and systems required to use the broadcast model in both sending tweets and reconstructing the timeline.

5.4.1 Channels

Our broadcast model will need two public shared communication channels. Public means that everything can read/see everything. Shared means that everybody can write to it.

The two channels can be implemented in different ways. One option is to give each communication channel a separate shared Twitter account, the other option is to use 'identifiers' (or message type or channel identifier) in the tweets, to separate the two channels within one account.

C_{msg} - Tweets (encrypted)
C_{key} - Public keys of users.

5.4.2 Setup

Before we can start tweeting and reconstruct the timeline in Pwitter, our method requires a setup phase. This process is illustrated for the user U and starts by asking the user for an encryption password. This password is used to make a symmetric key K_u. This key, K_u , the User-known Password-based Symmetric Key, is used to encrypt the settings of the Pwitter user and are stored within Twitter itself.

The settings of a user consist of at least the following fields: the list of n friends indicated by $\{F_1, F_2, \ldots F_n\}$ and the private key of the user $K^U_{private}$.

These settings are stored within Twitter by encoding the settings in an image. One of the ideas could be to store this image as the profile image. Other solutions might be to post a tweet containing this setting. Although requesting/searching for a specific post of the current user is a Type I *read*, this will not reveal a connection, because the search is done within the context of the user itself.

During the login process of the user, the settings-image is loaded (profile or from special message in the own timeline), decoded and decrypted with the use of the K_u. This is all done within the client application, out of sight of Twitter, and effectively hiding the list of friends.

In addition to the list of friends, a Pwitter user U will generate a private/public key pair during setup phase, called $K^u_{private}$ and K^u_{public}. The public key of this pair is announced to everybody in the key channel C_{key}. The private key is stored in the user settings, protected by the symmetric key U_k, as mentioned in the previous paragraph.

Finding the needed public keys has to be done carefully. As explained in section 5.2 we cannot use *read* operations of category I to retrieve the keys on a per friend basis. The only solution is to let the client download *(read all)* , all keys for all users and (optionally) store them locally, for effectiveness. New keys could be requested with a *read from date x* operation, which is a category II operation. Whenever the looking for a key of a specific user, this lookup should be done client-side, preventing the SNO from making any connections at all.

Setup Steps to setup Broadcast model for user U

1: $Password \leftarrow U$ chooses password.
2: K_u = MakeSymmetricKey($Password$)
3: Generate unique $K^u_{private}$ and K^u_{public}.
4: $B_{settings}$ = { $K^u_{private}$, F_1 ... F_n }
Bag of settings. Contains private key and list of friends.
5: B_{enc} = Enc($B_{settings}$, K_u)
6: I_1 = makeimg(B_{enc})
7: Upload I_1 as banner or profile image.
8: Send K^u_{public} on channel C_{key}.

5.4.3 Tweeting

Tweeting in the privacy enhanced layer, with the list of friends hidden from the SNO, is one of the key concepts of using Twitter. To be able to reach the hidden friends without linkability and using a public shared channel, we need an algorithm to perform this action.

We will illustrate our algorithm for the case that Alice A wants to tweet to her n number of hidden friends, called $F_1, F_2 ... F_n$ or F_A as list. The use of the public/private keys of Alice and her friends are indicated by the scheme K^A_{public} and $K^{F_1}_{public}$ etc.

Both symmetric and public/private cryptographic is used in our algorithm. A one-time-use symmetric session key $K_{session}$ is generated per Tweet, which is used to encrypt the message M_{org}. The main problem is however to distribute the session key to the specific friends of A, in a public shared channel C_{msg}, and considering that no direction communication (and OOB connections) is available, to prevent the linkabilty. Our algorithm solves this problem by adding an access-control-list L_{acl} to the final message. This access-control-list will contain the session key multiple times: for each friend the $K_{sesssion}$ is encrypted with the public key of the friend $K^{F_i}_{public}$, allowing restoring the session key to the friend F_i only.

In addition we add a hash H_{msg} to the final Tweet, which is calculated from the encrypted message M_{enc} and signed with the private key of Alice $K^A_{private}$). This hash authenticates the sender as we will show in the timeline reconstruction.

More formally, we can summarize our Tweeting algorithm as follows:

Tweet Steps to send a message from A to the broadcast channel C_{msg}

1: $M_{org} \leftarrow A$ composes a new message.
2: $K_{session} \leftarrow$ onetime symmetric key generated by A
3: $M_{enc} = Enc(M_{org}, K_{session})$
4: $I_1 = makeimg(M_{enc})$
Encodes the encrypted message into an image.
5: $H_{msg} = Enc(Hash(M_{enc}), K^A_{private})$
A hash is calculated and signed with the private key of the sender.
6: **for all** f in F_A **do**
7: $L_{acl} += Enc(K_{session}, K^f_{public})$
8: **end for**
9: $I_2 = makeimg(L_{acl})$
Encodes the access control list into an image.
10: Tweet on C_{msg} the H_{msg} and the images I_1 and I_2

Implementation remarks

Due to Twitters character limit of a single Tweet of 140 characters, we needed a solution to overcome this limitation. That's why the encrypted message and the access-control-list are encoded into two images.

5.4.4 Timeline

Timeline reconstruction means that we will get the messages from the broadcast channel and filter display only the relevant messages, decrypt them and show them in the GUI.

We will show the process of timeline reconstruction by using Bob B as our Pwitter user. Assuming that Bob is one of Alice A's friends F_A ($B \in F_A$) and Alice posted her encrypted message on the message channel C_{msg}.

The first step is to download all messages from the message channel C_{msg} , which are new since the last time, resulting in m new messages. Because this is a *read type II* operation, the SNO cannot derive any relationships from this action. The client can easily filter the messages because each sender S is (publicly) known and Bob's friends F_B are known to the client. This results in messages where each message contains: a sender S, the H_{msg} and two images I_1 and I_2. Image I_1 containing the encoded version of the original encrypted message and image I_2 the access control list, as explained in 5.4.3.

To verify the authenticity of the message, the senders public key K^S_{public}, is retrieved from the locally saved key list, as explained in 5.4.2 and used to decrypt the H_{msg}. This contains the calculated hash by S of the message, which is compared to the locally computed hash.

To be able to reconstruct the original message, we need to find the used $K_{session}$ from the access control list, which is encoded in image I_2. This list contains one row where Bob B could use his private key $K^B_{private}$ to decrypt a session key. We leave it up to the implementation to choose a suitable verification mechanism to check whether or not a decrypt results in a useful session key, by example: Prefix the session key, add checksum code, etc... The $K_{session}$ is used to decrypt the original message, encoded in image I_1. The result is added to the timeline.

More formally, we can summarize our Timeline reconstruction algorithm as follows:

Timeline Steps to reconstruct the timeline of user B with messages from C_{msg}

```
1: load newest messages from channel C_msg.
2: for all m in messages do
3:    get sender S, H_msg and images I_1, I_2 from message m.
4:    if S in F_B then
5:       M_enc = decodeimg(I_1)
6:       H_msg = Dec(H_msg, K^S_public)
7:       if (H_msg = Hash(M_enc) then
8:          L_acl = decodeimg(I_2)
9:          for all line in L_acl do
10:            K_session = try Dec(line , K^B_private)
11:         end for
12:         M_org = Dec(M_enc,K_session)
13:         send M_org to timeline
14:
15:      end if
16:   end if
17: end for
```

Complexity

We define m_{total} as the total number of new messages in the broadcast channel, m_U as the number of messages which are applicable to user U and finally f as the average number of friends any user would have (which is the number of lines in the ACL). With these definitions we can formulate the time complexity of this algorithm. Filtering the m_{total} messages to a list which only consists of the messages applicable to user U can be considered as a quick operation because the sender is directly visible in the message itself. The time complexity of the timeline reconstruction algorithm is $O(m_U * f)$.

In section 5.7.3 we discuss an idea that could lead to a time complexity of $O(m_U)$ in most cases by skipping the iteration of the access control list.

5.5 Broadcast II

In the design of our proof of concept, we chose to use receiver anonymity only. Although the rest of the protocol makes the connection between sender and receiver unlinkable, you can opt for a solution where the sender is hidden as well. For completeness we give some directions to hide the sender. One disadvantage of this approach is that timeline reconstruction is more inefficient, as we will explain in section 5.5.1.

When trying to hide both sender and receiver from the SNO, one should take extra care when sending messages on the broadcast channel and utilizing the API. The SNO could easily trace back the sender from matching IP address when using the two accounts (user and public) on the API.

5.5.1 Algorithm changes

The tweet and timeline algorithm as presented in section 5.4.3 and 5.4.4, needs some changes before the sender is hidden.

In this model, the sender will be included in the Image I_1 and is done in step 3 of the Tweeting algorithm: instead of encoding M_{org}, you encode $M_{org} + S$ with the $K_{session}$. Instead of using the sender account to tweet the message, as we did in the proof of concept, we need to use the broadcast account to place the tweet.

Timeline reconstruction is more challenging and less efficient in this variant of the broadcast model because the sender of the message is hidden. As a result, the receiver cannot determine which message in the broadcast channel is addressed to him and needs to try to decrypt every message.

The algorithm now starts with retrieving the session key $K_{session}$. If this succeeds then the encode message M_{enc} with sender S is restored and the rest is quite similar to the original algorithm. The complete timeline reconstruction in case of hidden sender is:

Timeline II Steps to reconstruct the timeline of user B with messages from C_{msg} in case of hidden sender

```
1: load newest messages from channel C_msg.
2: for all m in messages do
3:    get H_msg and images I_1, I_2 from message m.
4:    L_acl = decodeimg(I_2)
5:    for all line in L_acl do
6:       K_session = attempt Dec(line , K^B_private)
7:    end for
8:    if K_session not null then
9:       M_enc = decodeimg(I_1)
10:      M_org, S = Dec(M_enc,K_session)
11:      if S in F_B then
12:         H_msg = Dec(H_msg, K^S_public)
13:         if (H_msg = Hash(M_enc) then
14:            send M_org to timeline
15:         end if
16:      end if
17:   end if
18: end for
```

Complexity

Again we define m_{total} as the total number of new messages, m_U as the number of messages which are applicable to user U and finally f as the average number of friends any user would

have. With these definitions we can formulate the time complexity of this version of the timeline reconstruction algorithm.

Because the sender S is not visible in the message, we cannot apply fast filtering. For every single new message, the images are needed and a also scanning the ACL is required to try to recover the session key, which is needed to determine the sender of the message. The time complexity of this version of the algorithm is $O(m_{total} * f)$. Because $m_{total} >> m_U$ this is less efficient then the proof-of-concept version of the broadcast-model, where only receiver anonymity is preserved.

5.5.2 IP address

In the broadcast model, the sender is using two accounts. One of them is his original Twitter account which is used to request the user's normal timeline and recover the hidden storage images. The other account is built into the Pwitter client to connect to the broadcast channels. Without masking the used IP address, the SNO could simply deduct which sender tweeted in the broadcast channel, based on the same IP address of both accounts.

To hide the IP address from the SNO we propose to use Tor. Tor was introduced in 2004 by Dingledine [DMS04] as a so-called onion routing system. The idea of this kind of systems is to route IP traffic multiple times between nodes to hide the originating IP address. By using Tor when making API calls, the SNO cannot longer deduce a connection based on a new message in the broadcast channel and the IP addresses used by the user account and broadcast account.

5.6 Conclusion

As we showed above and in the proof of concept, a broadcast model could work for hiding your social network.

To summarize some of the advantages of the proposed algorithm, we can start by concluding it fulfills our main goal of hiding the social network for the operator while keeping basic Twitter functionality like timeline reconstruction and tweeting. Due to careful usage of the API, no transactions are done which could lead to the reconstruction of the friends by the SNO. The solution does not depend on a central server or storage, preventing it from becoming a SNO itself. The one-time used encryption keys in combination with key distribution per message, makes it (arguably) more secure than an OOB shared key strategy.

5.7 Future work

5.7.1 ACL size

In the proposed algorithm, the size of the access control list, would reveal the number of friends. To prevent the SNO from knowing this fact, some obfuscation could be applied, by adding fake friend items in the list or giving the list a fixed size.

5.7.2 Access control list

When trying to find the working session key $K_{session}$ in the access control list L_{acl}, one should be able to verify whether or not the decryption process resulted in a valid session key. One idea could be to concatenate $K_{session}$ with a known magic string and verify the existence of this magic string in the decrypted form. Another idea could be that the $K_{session}$ is extend with some sort of short-hash code.

5.7.3 Performance

One can assume that multiple messages are send from the sender S, before a change in the sender's list of friends F_S occur. In other words, sending messages happens more frequently then changing friends.

To prevent unnecessary scanning all the lines of the accces control list, we could store the last know 'line number' in the ACL of the sender in the friend-list of the user. When trying to find the correct ACL line , we should try the stored last-known working line number first. If this fails, a rescan of all the lines is necessary.

5.7.4 Efficient storage

Besides storing the ACL per message, one could opt for a publication of the ACL in the key channel C_{key}. In this case the key channel will have messages consisting of the not only the public key of a user, but will contain the ACL of that user as an image as well. This will save resending the ACL per messages, but might be less secure because the same 'session key' is reused again. Secondly : revoking a user from the friend list will result in generating a completely new ACL with new keys for all users.

Because the key channel C_{key} must be 'cached and stored' client side, to prevent *read type I* operations, the key channel should be as light as possible.

5.7.5 Weakness

The publication of the public keys K^U_{public} of a user U in in the key channel C_{key} is easily 'forged' , because a channel is a public channel. Some authentication has to be applied, otherwise impersonation is easily done. Ideas of using Pwitter as some sort of authority to validate keys, will result in Pwitter being a trusted party (and finally a SNO) itself, which was one of the main goals to prevent.

Another idea could be to publish the key not directly within the shared key channel, but to publish it on the users timeline. The Pwitter account follows the users, and a bot will retweet only key messages to the key channel of all users. This prevents 'users' from forging other users. However, it will not prevent Twitter to send forged key messages in this channel.

Chapter 6

Conclusion

6.1 Conclusion

This thesis aimed to explore to what extent it is possible to hide the user's social network from a social network operator. We hope to have contributed to ongoing research on improving the privacy of users of online socials networks, by proposing solutions to hide the social network from a social network operator.

We have designed two approaches that enable users to hide their social network from the SNO. The first approach is based on accounts that are used exclusively to communicate with another user. These '1-to-1' accounts are considered *relationship-pseudonyms* and are effective in hiding the social relation from the SNO. However, some practical limitations apply which makes this model more difficult to implement. The design of the second approach is quite different in nature and uses broadcasting to obtain recipient anonymity. This model has been implemented as a proof-of-concept and works in practice. Furthermore, a detailed description of the broadcast model and techniques has been conducted to find the most suitable method to make this approach work.

Next to the practical elaboration of the broadcast-model, we have modelled the accounts-model in the applied pi calculus. In order to do so, we have also formalised our notions of the social network and various privacy-related concepts. This provided the framework that allowed us to partly verify correctness of the tweeting process. The framework that has been developed for the formal verification should -in theory- also allow for the verification of other models that aim to hide the social network from an adversary.

We can conclude that there exists various possibilities to hide a user's social network from a social network operator. While we have elaborated two different approaches to do this, we certainly do not rule out that other approaches could work as well. Analysis of the two models revealed that the challenges merely are of practical nature, rather than fundamental. While this may sound easier to overcome, these practical challenges should not be underestimated. For the functioning of both models, scaling may have serious consequences.

Although Twitter was used as the OSN to build our proof-of-concept of the broadcast-model, the concepts and methods should be generally applicable to other OSNs as well. The only requirement is that a OSN has an API which the privacy-enhancing layer can interact with.

6.2 Research recommendations

We hope that this thesis will inspire other scholars to explore possibilities to hide the social network from social network operators by developing different models. Furthermore, we have identified various research directions that could improve the approaches chosen in this work. These will be briefly pointed out below.

As mentioned, both models may encounter problems when scaling. For example, the rate limit of the API may cause problems for the broadcast-model, since the Pwitter-client will have to process many requests as the user-base increases. To some extent, this may be solved by optimising the amount of the shared accounts and the position of these accounts within the entire social network graph. A more radical solution to obtain independence from limitations imposed by Twitter is by implementing this approach as a browser-plugin, rather than an application that uses the Twitter API. This approach has already been explored by [BBS+09].

In our models, we did not consider the way in which potential friends are found. Since simply searching friends could easily reveal a social relation, various researchers have proposed solutions to privately find friends [FN07, Mez11]. Our models could be extended by incorporating these techniques in the model.

The privacy of the social network could even further be improved if we could hide the mere fact that a privacy-enhancing layer is used. In combination with the approach to use a browser-plugin instead of the Twitter API, steganography techniques can be used to hide the (encrypted) messages or profile information. These techniques have already been proposed for profile information by [GTF08] and for messages by [BIČ+13].

The access control list, as presented in the methods of the broadcast-model, is a construction to overcome the limitations of most group based encryption methods. This could be more efficient by looking for methods that do not require this list and which essentially would solve the key establishment procedure of a encryption method with respect to the requirements like: unlinkable communication and dynamic group membership.

Lastly, we have only partly verified one of the protocols using a formal verification language. We have pointed out there are multiple possible ways in which an OSN may obtain information about the social network, such as linking messages. Further work should be conducted to verify these other possible attacks as well. We also have only verified the tweeting process, which is arguably the easier part of the accounts-model. The more complex relation establishment part of the protocol deserves to be formally verified as well, since the complexity indeed makes it more likely that it contains weaknesses that are overlooked when it is only informally analysed.

Appendices

Appendix A

Research planning

A.1 Time schedule

Our time schedule is based on 20 weeks with 20 hours per week each with one or more activities planned in blocks of 2 weeks (see table A.1). Some activities will run in parallel. The defined main activities (and their lead time) are:

- General (2 weeks)
  Startup, introduction and meetings
- Proposal (4 weeks)
  Writing the Research Proposal;
- Models (8 weeks)
  Research on the models: introduction, formal description, design, scalability, security analysis, limitations, conclusion;
- Peer (1 week)
  Peer presentation preparation;
- Implementation (4-6 weeks)
  Implementation of one of the models; code environment setup
- Thesis (6-8 weeks)
  Combining research proposal, research on the models and implementation notes. Adding conclusions and general parts. Process feedback and reviews.

A.2 Testing and documentation

Due to the scope of this project, together with the tight time schedule, it is likely that this project will be considered a 'proof of concept', rather than of a finished product. Unit tests and automated integration tests will be written where possible. Source code will be commented correctly and documented. Of course, a detailed system overview of all concepts is provided as well in the corresponding documentation.

Table A.1: Planning

week 36,37	General
week 38,39	General; Proposal
week 40,41	Proposal
week 42,43	Proposal
week 44,45	Models
week 46,47	Models; Peer;
week 48,49	Models; Peer; Implementation
week 50,51	Models; Implementation; Thesis
week 52,1	Free
week 2,3	Thesis; Implementation
week 4,5	Thesis; Implementation
week 6,7	Thesis

A.3 Tools

Within this project we will use the following tools:

- Git[1]
  Git is a distributed source control system, which will be used for maintaining the sourcecode of our project.
- ShareLatex[2]
  ShareLatex is an online Latex editing and generating tool, which will be used for writing the documents.
- Slack[3]
  Day to day communication is done using Slack. It is a real time messaging system specialized for (remote)team work.
- Mendeley[4]
  This tool is used for managing research papers and organising references.

[1] `https://git-scm.com`
[2] `https://www.sharelatex.com`
[3] `https://www.slack.com`
[4] `https://www.mendeley.com`

A.4 Risks

Various risks can be identified:

1. Scope: Twitter allows a lot of different communication methods (hashtag, direct messages, user tagging, retweeting). Adding a privacy layer may require a lot combinatorial complexity to get the messaging system between Private Tweeting-users and regular users working. This risk is minimised by narrowing the scope of the project to users that use 'Protected Tweets'.

2. Legality: by using Twitter, the user agrees to comply with various terms and conditions. Since it is possible that hiding the user's network, encrypting messages and creating 'fake' accounts could be a violation of these terms and conditions, close examination of the extent to which this project is compliant with Twitter's terms and conditions is necessary.

A.5 Quality management

Since this project aims to develop a proof of concept, quality of the product is mainly dependent of correctly implementing core functionality. This is ensured by having a sound underlying formal framework, which will be developed in the individual research parts.

The quality of the process is ensured by having frequent meetings with the project owner and open communication lines between team members.

A.6 Communication

Communication within the team:

- weekly Skype meetings
- daily chats using Slack (optional)

Communication between team and the project owner

- Every 40 effective hours a Skype meeting will be planned with the project owner, resulting in a Skype meeting every 2 weeks.
- Notes are taken during meetings and send by mail to the team and the project owner.

A.7 Architecture

Because the development of the architecture is dependent on the chosen model, these will be developed within the individual research parts.

Appendix B

Applied pi calculus

B.1 Syntax and Informal Semantics

In the applied pi calculus, there is an infinite set of *names*, and infinite set of *variables* and a *signature* Σ, which contains a finite set of function symbols. Names are used for communication channels and other constants and function symbols are used to define the terms, defined by the grammar:

$U, V ::=$	terms
$a, n, ...$	name
$x, y, ...$	variable
$f(U_1, ..., U_l)$	function application

A function application f ranges over the function symbols of Σ with arity l. All of the terms are used to describe the processes. Plain processes are similar to processes in the pi calculus, but unlike the pi calculus can also contain terms. They are denoted:

$P, Q, R ::=$	processes (or plain processes)
0	null process
$P \mid Q$	parallel composition
$!P$	replication
$\nu n.P$	name restriction
if $U = V$ then P else Q	conditional
$u(x).P$	message input
$\bar{u}\langle M\rangle.P$	message output

A plain process can be the null process 0, which does nothing; two subprocesses P and Q can in run in parallel, denoted $P \mid Q$; the replication $!P$ means that an infinite number of copies of P are running in parallel; $\nu n.P$ denotes the restriction of n in P; the process $u(x).P$ is waiting

for input on channel u, where the incoming message will can be referenced with x in P; finally, $\bar{u}\langle M\rangle.P$ is ready to transmit message M on channel u. Processes are extended with *active substitutions*:

$A, B, C ::=$	extended processes
P	plain process
$A \mid B$	parallel composition
$\nu n.A$	name restriction
$\nu x.A$	variable restriction
$\{^M/_x\}$	active substituion

An active substitution $\{^M/_x\}$ is a process that has previously output M, and replaces every variable x with the term M in the process it comes into contact with. This can be restricted: $\nu x.(\{^M/_x\} \mid P)$, which corresponds to "let $x = M$ in P". Names and variables are free if they are not defined by restriction and inputs. We denote the sets of free variables, free names, bound variables and bound names of a process A as respectively $\mathrm{fv}(A)$, $\mathrm{fn}(A)$, $\mathrm{bv}(A)$ and $\mathrm{bn}(A)$.

An extended process A can be mapped to its frame $\varphi(A)$ using active substitutions by parallel composition and restriction. We let φ and ϕ range over frames. A frame $\varphi(A)$ can be seen as an approximation of A that accounts for the static knowledge exposed by A to its environment. The domain $dom(\varphi)$ of a frame φ is the set of variables that φ exports.

B.2 Semantics

The signature Σ is equipped with an equational theory E. This is a set of equations of the form $M = N$, where $M, N \in T_\Sigma$. The operational semantics are defined by two relationships: *structural equivalence* and *internal reduction*. The processes may run within a context $C[\cdot]$, which can be used to represent an adversarial environment. It is an extended process with a hole, which can be filled with an extended process A. This results in $C[A]$. An evaluation context is a context whose hole is not under a replication, a conditional, an output, or an input. Structural equivalence is noted $\equiv$ and formally is the smallest equivalence relation on extended processes that is closed by α-conversion on both names and variables, such that:

$$A \equiv A \mid 0 \tag{B.1}$$
$$A \mid (B \mid C) \equiv (A \mid B) \mid C \tag{B.2}$$
$$A \mid B \equiv B \mid A \tag{B.3}$$
$$!P \equiv P \mid !P \tag{B.4}$$

$$\nu n.0 \equiv 0 \tag{B.5}$$
$$\nu u.\nu w.A \equiv \nu w.\nu u.A \tag{B.6}$$
$$A \mid \nu u.B \equiv \nu u.(A \mid B) \quad \text{where } u \notin \mathrm{fv}(A) \cup \mathrm{fn}(A) \tag{B.7}$$

$$\nu x.\{^M/_x\} \equiv 0 \tag{B.8}$$
$$\{^M/_x\} \mid A \equiv \{^M/_x\} \mid A\{^M/_x\} \tag{B.9}$$
$$\{^M/_x\} \equiv \{^N/_x\} \quad \text{where } \Sigma \vdash M = N \tag{B.10}$$

Internal reduction is noted $\rightarrow$ and is the smallest equivalence relation in extended processes under structural equivalence and application of evaluation contexts such that:

$$\bar{a}\langle M\rangle.P \mid a(x).Q \rightarrow P \mid Q\{^M/_x\} \tag{B.11}$$
$$\text{if } N = N \text{ then } P \text{ else } Q \rightarrow P \tag{B.12}$$
$$\text{if } L = M \text{ then } P \text{ else } Q \rightarrow Q \quad \text{where } \Sigma \nvdash M = N \tag{B.13}$$

This means that a process can execute without interacting with its environment by evaluating if-statements or if internal sub-processes communicate with each other.

Internal reductions are extended by *labelled reduction* ($\xrightarrow{\alpha}$), which enables us to reason about the interaction with environment. The label α can either be an input or the output of a channel name or variable of base type. The following rules are used:

$$a(x).P \xrightarrow{a(M)} P\{^M/_x\} \tag{B.14}$$
$$\bar{a}\langle u\rangle.P \xrightarrow{\bar{a}\langle u\rangle} P \tag{B.15}$$
$$\frac{A \xrightarrow{\bar{a}\langle u\rangle.P} A' \quad u \neq a}{\nu u.A \xrightarrow{\nu u.\bar{a}\langle u\rangle} A'} \tag{B.16}$$
$$\frac{A \xrightarrow{\alpha} A' \quad u \text{ does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'} \tag{B.17}$$
$$\frac{A \xrightarrow{\alpha} A' \quad bv(\alpha) \cap fv(B) = bn(\alpha) \cap fn(B) = 0}{A \mid B \xrightarrow{\alpha} A' \mid B} \tag{B.18}$$
$$\frac{A \equiv B \quad B \xrightarrow{\alpha} B' \quad B' \equiv A'}{A \xrightarrow{\alpha} A'} \tag{B.19}$$

We will write $\stackrel{\alpha}{\Rightarrow}$ for an arbitrary number of internal reductions, followed by a labelled reduction and an arbitrary number of internal reductions [ACRR10].

B.3 Equivalences

As pointed out in the introduction, it is common to prove some kind of *observational equivalence* between processes when formalising privacy in applied pi. Its definition as given by [AF01] is: we write $A \Downarrow a$ when A emits on channel a, that is, when $A \to^* C[\bar{a}\langle M\rangle.P]$ for some evaluation context C that does not bind a.

Definition 12 ([AF01])**.** *Observational equivalence ($\approx$) is the largest relation between closed extended processes with the same domain such that $A \mathcal{R} B$ implies:*

1. *if $A \Downarrow a$ then $B \Downarrow a$;*
2. *if $A \to^* A'$ then $B \to^* B'$ and $A' \mathcal{R} B'$ for some B';*
3. *$C[A] \mathcal{R} C[B]$ for closing evaluation contexts C.*

Furthermore, *labelled bisimilarity* ($\approx_l$) can be used, because it coincides with observational equivalence but is easier to reason with. Labelled bisimilarity uses *static equivalence* $\approx_s$:

Definition 13 ([AF01])**.** *Closed frames φ and ϕ are statically equivalent, $\varphi \approx_s \phi$, if*

1. *$dom(\varphi) = dom(\phi)$;*
2. *$\forall$ terms M and N, $(M =_E N)$ in $\varphi \Leftrightarrow (M =_E N)$ in ϕ.*

Definition 14 ([AF01])**.** *Labelled bisimilarity ($\approx_l$) is the largest symmetric relation $\mathcal{R}$ on closed extended processes with the same domain such that $A \mathcal{R} B$ implies:*

1. *$A \approx_s B$;*
2. *if $A \to A'$ then $B \to^* B'$ and $A' \mathcal{R} B'$ for some B';*
3. *if $A \stackrel{\alpha}{\to} A'$ and $fv(\alpha) \subseteq dom(A)$ and $bn(\alpha) \cap fn(B) = 0$, then $B \stackrel{\alpha}{\Rightarrow} B'$ and $A' \mathcal{R} B'$ for some B'.*

Appendix C

ProVerif scripts

C.1 Normal tweeting process

(Declarations *)*

type tweet.
type relation.
type id.

fun create_relation(id, id): relation.
reduc forall s: id, f: id; subject(create_relation(s, f)) = s.
reduc forall s: id, f: id; follows(create_relation(s, f)) = f.

fun create_tweet(id, bitstring): tweet.
reduc forall s: id, m: bitstring; message(create_tweet(s, m)) = m.
reduc forall s: id, m: bitstring; sender(create_tweet(s, m)) = s.

free read, send, login, req:channel [*private*].
free c, f, r, t:channel.

free alice, bob:id [*private*].

(queries *)*

query *attacker*(create_relation(alice, bob)).

event tweetReceived(tweet).
query tw:tweet; event(tweetReceived(tw)).

(processes *)*

let U(i:id, i':id) =
    out(login, create_relation(i, i')) |
    !**new** m:bitstring; **out**(send, create_tweet(i, m)) |
    !**out**(req, i) |

```
        !in(read, w: tweet); event tweetReceived(w).

let I() =
        in(login, x':relation); out(f, x') |
        in(send, z':tweet); out(c, z') |
        in(req, y':id); out(r, y') |
        in(t, w':tweet); out(read, w').

let B() =
        in(f, x:relation);
        in(r, y:id);
        in(c, z:tweet);
        if (subject(x) = y ∧ follows(x) = sender(z)) then out(t, z).

process
        U(alice, bob) |
        U(bob, alice) |
        !I |
        !B
```

C.2 Private tweeting process

```
(* Declarations *)

type tweet.
type key.
type relation.
type id.

fun enc(key, tweet): bitstring.
reduc forall m: tweet, k: key; dec(k, enc(k, m)) = m.

fun create_relation(id, id): relation.
reduc forall s: id, f: id; subject(create_relation(s, f)) = s.
reduc forall s: id, f: id; follows(create_relation(s, f)) = f.

fun create_tweet(id, bitstring): tweet.
reduc forall s: id, m: bitstring; message(create_tweet(s, m)) = m.
reduc forall s: id, m: bitstring; sender(create_tweet(s, m)) = s.

free read, send, login, req:channel [private].
free c, f, r, t:channel.

free alice, bob:id [private].
free k:key [private].

(* queries *)
```

```
query attacker(create_relation(alice, bob)).

event tweetReceived(tweet).
query tw:tweet; event(tweetReceived(tw)).

(* processes *)

let U(i:id, i':id) =
        out(login, create_relation(i, i')) |
        !new m:bitstring; out(send, create_tweet(i, m)) |
        !out(req, i) |
        !in(read, w: tweet); event tweetReceived(w).

let I'(fake_i:id, fake_i':id, k:key) =
        in(login, x':relation); out(f, create_relation(fake_i, fake_i')) |
        in(send, z':tweet); out(c, create_tweet(fake_i, enc(k, z'))) |
        in(req, y':id); out(r, fake_i) |
        in(t, w':tweet); out(read, dec(k, message(w'))).

let B() =
        in(f, x:relation);
        in(r, y:id);
        in(c, z:tweet);
        if (subject(x) = y ∧ follows(x) = sender(z)) then out(t, z).

process
        U(alice, bob) |
        U(bob, alice) |
        !new fake_alice:id; new fake_bob:id; (I'(fake_alice, fake_bob, k) |
        I'(fake_bob, fake_alice, k)) |
        !B
```

Appendix D

Implementation

D.1 Introduction

In our attempt to hide to social network from the SNO, we presented two models. One of these models, the broadcast-model, is implemented as a proof of concept.

The software layer must have some specific characteristics to achieve our goals. One of them is the prevention of becoming a SNO itself where users should trust another party. No central database or storage should be allowed. This is achieved by storing the needed userdata encrypted within Twitter itself, as explained in 3.1.1. The proposed software layer in this paper is a separate application, rebuilding some of the Twitter interface site elements. In the rest of this chapter we will call it the Pwitter client.

The application should also be open source and run client-side to be as transparent as possible. These requirements pose some challenges and may compromise the usability. Nevertheless, they are considered necessary because the primary goal is to implement a privacy enhancing layer.

D.2 Technical design considerations

The goal is to achieve a stand alone software package which looks and operates the same like Twitters website. Because we are building a website in fact, it implies designing and building the frontend in HTML.

An additional requirement is that we do not want to use a central server where private information could be stored. To achieve this, one would need to run the application on a desktop. Presenting and interacting with HTML content in the browser requires a webservice, which is typically not available on a desktop device. To conclude we try building the tool as cross platform as possible.

These set of requirements and wishes resulted in the selection of our implementation technology stack of NodeJS and HTML5.

NodeJS is a platform that allows the execution of applications written in Javascript. It is typically used as a webserver. An advantage of this engine is that is available for the most popular plaforms.

D.3 Language

Because Javascript is used for the frontend (HTML5) and backend (NodeJS), we had a look at the different 'flavours' of Javascript that are currently available.

Due to the nature of Javascript as a loosely typed and functional language, people looked for solutions to introduce a more object oriented and a more Type-strict way to deal with the problems (like bugs introduced by loosely typing). This resulted in 'new' languages, on top of Javascript. On top means that the new languages are not compiled directly, ready for execution, but transpiled (source to source) back to Javascript. Some examples of these languages are: Coffeescript, ELM, Dart, Typescript. We choose Typescript as our Javascript flavour because it adds type checking and object oriented approach and it allows for the use of new Javascript features (ECMAscript 6), not available to all browsers.

D.4 Frontend GUI

Our GUI of Pwitter is based on the Twitter website itself. We want the users of Pwitter, to get the feeling that they can use Pwitter as easy as Twitter. To get started with the layout and CSS to design this front-end, we used Bootstrap CSS for the technical implementation of the layout and Twitter website for the look and feel. Bootstrap is a set of CSS (ironically made by Twitter), to kickstart a HTML project with good fundamental layout options, like their so-called grid system, to easily define regions on a page.

D.5 Frontend Framework

Building a desktop like application in HTML technology requires a responsive, flicker free web application. Single page applications (SPA) fullfill this requirement by loading only one HTML page, and after the initial loading, dynamically alter the content based on the current state and fetched data, without refreshing, reloading a complete new webpage from the server. This results in a 'desktop' like experience to the user.

There exists quite a few SPA frameworks (Ember, Backbone, React, AngularJS) and more are coming every couple of months. Every framework has his own advantages and disadvantages, which would take a separate paper to compare all of them. We chose Angular2 as SPA framework, mainly because it embraces Typescript as its preferred Javascript flavour.

Appendix E

Reflections

E.1 Team reflection

Planning Our initial timeline was a little bit optimistic, especially due to the fact that it was based on a smaller scope of the project. While writing the research proposal we discovered some interesting research areas and topics that we wanted to include in the thesis as well. This resulted in more work then initially estimated, but we believe the result was worth the extra effort.

Communication To discuss our progress and ideas, we scheduled regular meetings. Approximately every 2-3 weeks there was a teleconference call with Hugo Jonker, to present our work and get his feedback and recommendations. At the end of these calls we defined new goals for the next meeting and scheduled a new appointment right away. Having frequent meetings worked quite well in general, although there was sometimes little time to really dive into certain subjects. In the mean time we used Skype and Slack to communicate mutually. Slack is a chat messaging system that helped us to communicate in the periods we both worked on different days. Once in a while we planned a full day to meet and work together on writing and developing.

Collaboration Splitting the amount of work went quite natural and without struggles. Bas had more experience in writing thesis and Lucas had more experience in developing and implementation, although we used our background only occasionally for the division of tasks. Eventually we both wrote and developed, since we both wanted to learn something as well.

Tools We found a good set of tools to collaborate. An important one was Sharelatex, a online latex editor, which was used to write this document and helped a lot to actual work together, read, review and comment on each other work. Setting up a good file structure for the project to split out different parts of the thesis is advisable and we can highly recommend the tool to other teams.

Development For development we used a combination of tools that supported different platforms (Mac and Windows). Git was an excellent choice for our source control and facilitated co-developing while maintaining a personal copy of code during development, which resulted in a friction free development process. Visual Studio Code was our choice of IDE because it was support in both platforms. Although we developed cross-platform we didn't experience much complications with it. During development we started by splitting the work in front-end and

back-end actions. This method worked good in the initial phase of the project and when most of the general structure was in place, we divided the rest of the work per functionality/use-case.

Writing During our initial meeting with Hugo, he expressed his desire to have a more academic approach for this project, which meant the focus would be on the thesis rather than the software. This was fine, although the framework that the project needs be delivered in (and according to which it is examined), is more geared towards a 'software-development'-approach. Sometimes it was a bit of a struggle to fit these different requirements together. We realise that this may be the consequence of the transitional phase that was just taking place when we started the project, but nevertheless

E.2 Reflection Lucas

This bachelor computer science project started with an introduction form to introduce ourselves to the other students. One of the questions was: "What would you like to learn?".

In my daily work I have quite a lot experience in coding and developing applications but writing a thesis was something new for me. I really wanted to learn writing. It took some time to get started, especially the first sentences of every new chapter are still a little hard. Starting with the outline and the big picture was very helpful. Also my partner Bas and his seemingly effortless writing skills were inspiring to continue and once started, it was a very satisfying activity.

I was very happy to work on the broadcast-model as an alternative way to hide a social network from the social network operator. The most challenging part was to look at the problem in a more theoretical way. This was needed to apply it in a general sense and less as an concrete implementation in Twitter alone. This forced me to take some distance of the case and helped to write down the solution in more general terms and notation.

Communication with Bas and Hugo went very well. The tools we found like: ShareLatex for writing the thesis, Slack for chat communication and regular Skype meetings with Hugo, helped a lot during the project. It was also my first (real) experience with Latex as a typesetting system and like the way it helps concentrating on the content instead of the layout. Most challenging parts were the construction of the diagrams, the flows and algorithm descriptions to get expressed in Latex as well.

I would like to thank Hugo Jonker and Greg Alpar for their assistance and their patience during the process of completing this thesis.

E.3 Reflection Bas

While the process of writing a thesis always has its ups and downs, I'm quite happy with how things progressed. This is especially due to the good cooperation with Lucas. I think we complemented each other with our different skillsets quite well, as Lucas has a lot of experience with programming and I have more experience with writing. This led to a natural division of labour, while we both had the possibility to learn from each other's expertise. The fact that we quickly found a good set of tools that facilitated collaboration for both writing (ShareLatex, Mendeley),

programming (GitLab, VisualStudio Code) and communication (Slack, Skype), definitely helped that collaboration.

Writing a thesis can be a challenging process, but practice helps. Although the amount of writing practice in the bachelor was quite limited, this was quite the opposite in my previous degree of criminology. The writing skills that I learned there helped me a lot in the process of writing this thesis as well. My experience in programming is limited to what we have learned during the bachelor, but since Lucas was much more experienced in programming, this did not cause any difficulties. Furthermore, Hugo made clear during our first meeting that he preferred a more academic approach, where the thesis should be of high quality and implementation could more of a 'proof-of-concept', than a detailed implementation. This makes sense for this subject, since the practical value of the tool that we have designed is considerably less than the academic value.

The most challenging aspect of this thesis turned out to be the individual part of the thesis, which was modelling one of the protocols in the applied pi calculus. The formal modelling of security protocols is not in the curriculum of the bachelor, which meant that a lot of extra time was needed to get familiar with the subject. The fact that this turned out to be the most challenging part was not really a surprise, as Hugo warned me that it would be. Nevertheless, I am quite happy that I decided to take that challenge for two reasons. First of all, I feel that I learned a lot extra about modelling security protocols in formal languages such as the applied calculus. I believe that choosing an easier subject would not forced me to really dive into the material as was necessary for this subject. Secondly, I'm quite to happy to have learned more about this particular subject as I'm specialising in information security. Understanding the process of formal modelling and verification is valuable skill for that area of expertise.

Bibliography

[ACRR10] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proceedings - IEEE Computer Security Foundations Symposium*, pages 107–121, 2010.

[ADBS09] Jonathan Anderson, Claudia Diaz, Joseph Bonneau, and Frank Stajano. Privacy-enabling social networking over untrusted networks. In *Proceedings of the 2nd ACM workshop on Online social networks - WOSN '09*, page 1. ACM, 2009.

[AF01] Martin Abadi and Cedric Fournet. Mobile values, new names, and secure communication. *ACM SIGPLAN Notices*, 36(3):104–115, 2001.

[AG99] Martín Abadi and Andrew D Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, 1999.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[BBS+09] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication - SIGCOMM '09*, page 135. ACM, 2009.

[BIČ+13] Filipe Beato, Iulia Ion, Srdjan Čapkun, Bart Preneel, and Marc Langheinrich. For some eyes only. In *Proceedings of the third ACM conference on Data and application security and privacy - CODASPY '13*, page 1, New York, New York, USA, 2013. ACM Press.

[BJE+12] Michael Beye, Arjan Jeckmans, Zekeriya Erikin, Pieter Hartel, Reginald Lagendijk, and Qiang Tang. Privacy in online social networks. In *Computational Social Networks*, pages 87–113. Springer, 2012.

[Bla04] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *Proceedings - IEEE Symposium on Security and Privacy*, volume 2004, pages 86–100, 2004.

[Bla09] Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[BMP11] Michael Backes, Matteo Maffei, and Kim Pecina. A Security API for Distributed Social Networks. In *Ndss*, volume 11, pages 35–51, 2011.

[BS11] W Lance Bennett and Alexandra Segerberg. Digital media and the personalization of collective action: Social technology and the organization of protests against the global economic crisis. *Information Communication and Society*, 14(6):770–799, 2011.

[CFP07] Barbara Carminati, Elena Ferrari, and Andrea Perego. Private Relationships in Social Networks. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 163–171. IEEE, 2007.

[CGC11] Grainne Conole, Rebecca Galley, and Juliette Culver. Frameworks for understanding the nature of interactions, networking, and community in a social networking site for academic practice. *International Review of Research in Open and Distance Learning*, 12(3):119–138, 2011.

[CGI+99] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Proceedings - IEEE INFOCOM*, volume 2, pages 708–716, 1999.

[CMS09a] Leucio Antonio Cutillo, Refik Molva, and Thorsten Strufe. Privacy preserving social networking through decentralization. In *WONS 2009 - 6th International Conference on Wireless On-demand Network Systems and Services*, pages 145–152, 2009.

[CMS09b] Leucio Antonio Cutillo, Refik Molva, and Thorsten Strufe. Safebook: A privacy-preserving online social network leveraging on real-life trust. *IEEE Communications Magazine*, 47(12):94–101, 2009.

[Cre06] Cas Cremers. *Scyther: Semantics and verification of security protocols.* {Ph.D.} dissertation, Eindhoven University of Technology, 2006.

[CX10] Tang Chenxing and Wang Xiaodong. Preserving Privacy in Social Networks Against Subgraph Attacks. In *Social Networks*, pages 154–158. IEEE, 2010.

[DF07] Josep Domingo-Ferrer. A Public-Key Protocol for Social Networks with Private Relationships. In *International Conference on Modeling Decisions for Artificial Intelligence*, pages 373–379. Springer, 2007.

[DFVSGN08] Josep Domingo-Ferrer, Alexandre Viejo, Francesc Sebé, and Úrsula González-Nicolás. Privacy homomorphisms for social networks with private relationships. *Computer Networks*, 52(15):3007–3016, 2008.

[DJP13] Naipeng Dong, Hugo Jonker, and Jun Pang. Enforcing Privacy in the Presence of Others: Notions, Formalisations and Relations. In *Proc.\ 18th European Symposium on Research in Computer Security (ESORICS'13)*, volume 8134 of *LNCS*, pages 499–516. Springer-Verlag, 2013.

[DKR06] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Proceedings of the Computer Security Foundations Workshop*, volume 2006, pages 28–39, 2006.

[DKR09] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[DRS08] Stéphanie Delaune, Mark Ryan, and Ben Smyth. Automatic verification of privacy properties in the applied pi calculus. In *IFIP International Federation for Information Processing*, volume 263, pages 263–278, 2008.

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[FA03] Cédric Fournet and Martín Abadi. Hiding Names: Private Authentication in the Applied Pi Calculus. *Proc. Mext-NSF-JSPS international conference on Software security: theories and systems*, 2609:317–338, 2003.

[FBFF12] Ariel J Feldman, Aaron Blankstein, Michael J Freedman, and Edward W Felten. Social networking with frientegrity: privacy and integrity with an untrusted provider. In *Proceedings of the 21st USENIX conference on Security symposium*, volume Vol. 12, pages 31–31. USENIX, 2012.

[FN01] Amos Fiat and Moni Naor. Advances in Cryptology — CRYPTO 2001. *Lecture Notes in Computer Science*, 2139:480491, 2001.

[FN07] Michael J Freedman and Antonio Nicolosi. Efficient Private Techniques for Verifying Social Proximity. In *International Workshop on Peer-to-Peer Systems*, page 1, 2007.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 89–98, New York, NY, USA, 2006. ACM.

[Gra16] Martin Grandjean. A social network analysis of Twitter: Mapping the digital humanities community. *Cogent Arts & Humanities*, 3(1):1171458, 2016.

[GTF08] Saikat Guha, Kevin Tang, and Paul Francis. Noyb. In *Proceedings of the first workshop on Online social networks - WOSP '08*, page 49, New York, New York, USA, 2008. ACM Press.

[HS02] Dani Halevy and Adi Shamir. The lsd broadcast encryption scheme. In *Annual International Cryptology Conference*, pages 47–60. Springer, 2002.

[KHMS11] Jan Kietzmann, Kristopher Hermkens, Ian Paul Mccarthy, and Bruno S Silvestre. Social Media? Get Serious! Understanding the functional building blocks of social media. *Business horizons*, 54(May):241–251, 2011.

[KR05] Steve Kremer and Mark D. Ryan. Analysis of an electronic voting protocol in the applied pi calculus. *Programming Languages and Systems*, pages 186–200, 2005.

[LB08] Matthew M. Lucas and Nikita Borisov. FlyByNight. In *Proceedings of the 7th ACM workshop on Privacy in the electronic society - WPES '08*, page 1, New York, New York, USA, 2008. ACM Press.

[Mez11] Ghita Mezzour. Privacy-Preserving Path Discovery In Social Networks. In *International Conference on Cryptology and Network Security*, pages 189–208. Springer, 2011.

[Mor11] Evgeny Morozov. *The Net Delusion*. PublicAffairs, 2011.

[MVdV04] S. Mauw, J. Verschuren, and E.P. de Vink. A formalization of anonymity and onion routing. *Proceedings of ESORICS 2004*, pages 109–124, 2004.

[NNL01] Dalit Naor, Moni Naor, and Jeff Lotspiech. Revocation and tracing schemes for stateless receivers. In *Annual International Cryptology Conference*, pages 41–62. Springer, 2001.

[NS09] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *Proceedings - IEEE Symposium on Security and Privacy*, pages 173–187. IEEE, 2009.

[PH10] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management. *Technical University Dresden*, pages 1–98, 2010.

[PRC14] Sai Teja Peddinti, Keith W. Ross, and Justin Cappos. "On the internet, nobody knows you're a dog". In *Proceedings of the second edition of the ACM conference on Online social networks - COSN '14*, pages 83–94. ACM, 2014.

[RS11] Mark D. Ryan and Ben Smyth. Applied pi calculus. *Cryptology and Information Security Series*, 5:112–142, 2011.

[SSN+10] Seok-Won Seong, Jiwon Seo, Matthew Nasielski, Debangsu Sengupta, Sudheendra Hangal, Seng Keat Teh, Ruven Chu, Ben Dodson, and Monica S. Lam. PrPl. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services Social Networks and Beyond - MCS '10*, MCS '10, pages 1–8, New York, NY, USA, 2010. ACM.

[SZF10] Jinyuan Sun, Xiaoyan Zhu, and Yuguang Fang. A privacy-preserving scheme for online social networks with efficient revocation. In *Proceedings - IEEE INFOCOM*, 2010.

[TKvdV15] Loes Turpijn, Samantha Kneefel, and Neil van der Veer. Nationale Social Media Onderzoek. Technical report, Newcom Research & Consultancy B.V., 2015.

[TSGW09] Amin Tootoonchian, Stefan Saroiu, Yashar Ganjali, and Alec Wolman. Lockr. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies - CoNEXT '09*, page 169, New York, New York, USA, 2009. ACM Press.

[ZSZF10] Chi Zhang, Jinyuan Sun, Xiaoyan Zhu, and Yuguang Fang. Privacy and Security for Online Social Networks: Challenges and Opportunities. *IEEE Network*, 24(4):13–18, 2010.