ABSTRACT

It is well known that online social networking sites (OSNs) such as Facebook pose risks to their users' privacy. OSNs store vast amounts of users' private data and activities and therefore subject the user to the risk of undesired disclosure. The regular non-tech-savvy Facebook user either has little awareness of his privacy needs or is not willing or capable to invest much extra effort into securing his online activities.

In this paper, we present a non-disruptive and easy-to-use service that helps to protect users' most private information, namely their private messages and chats against the OSN provider itself and external adversaries. Our novel Confidentiality as a Service paradigm was designed with usability and non-obtrusiveness in mind and requires little to no additional knowledge on the part of the users. The simplicity of the service is achieved through a novel trust splitting approach integrated into the Confidentiality as a Service paradigm. To show the feasibility of our approach we present a fully-working prototype for Facebook and an initial usability study. All of the participating subjects completed the study successfully without any problems or errors and only required three minutes on average for the entire installation and setup procedure.

1. INTRODUCTION

Social networks currently play an important role in many people's daily lives. The amount of social interaction taking place on the Internet is growing rapidly and benefits from new technology. In 2011, around 2 billion people around the world were using Social Networks (SNs), such as Facebook (750 million users)\(^1\). However, this also poses a risk to a user's information in terms of privacy and trust \(^5\). Studies have shown that users value their privacy in general, but act the opposite in social networks \(^1\)\(^1\)\(^4\)\(^7\).

Previous work has proposed measures to improve privacy concerning unwanted disclosure to other SN users (e.g. \(^25\)\(^4\)\(^12\)\(^10\)). While the privacy implications of publicly sharing information is slowly finding its way into the users' minds, the problem of giving the SN provider all their (possibly highly private) information without any means of control has not been properly recognised by the general public yet.

Generally, the business model of SN providers (make money with advertisements based on the users' content) collides with the users' need for privacy. However, this is so easily forgotten over flashy games and catchy features. It is only in Facebook's interest to protect its users' information for publicity reasons. While this might be acceptable for many users in principle, we believe that there should be a clear distinction between content that obviously is intended to be public or at least semi-public (e.g. wall posts, likes, or fan site memberships) and information that is thought to be personal and confidential (e.g. private messages and chats). Semi-public content does not need to be protected to the same extent as private content, since its purpose is to be consumable by a higher number of users. For private content, on the other hand, we argue that there needs to be a solution to effectively protect a user's privacy, even from the SN providers themselves, without burdening the users with security related workflows or requiring specialised knowledge.

Lucas et al. \(^18\) actually argue that the current legal framework in the U.S. does not protect a user's privacy at all when interacting with Facebook, because the users have "no reasonable expectation of privacy" when using the service. Additionally, since Facebook is based in Menlo Park, Ca, USA, all the users' data stored on Facebook's servers is subject to the US Electronic Communications Privacy Act\(^2\). Hence, there is a need for external digital measures to pro-

---

\(^1\)http://www.socialnomics.net/2011/08/16/social-network-users-statistics/

\(^2\)http://cpsr.org/issues/privacy/ecpa86/
tect a user’s privacy, which has been recognised in several previous approaches [18, 3, 4, 17, 11, 2].

While the idea of cryptographically securing messages and other SN content is not a novelty, previous approaches often suffer from a lack of usability by impeding the user’s regular workflow. Especially because privacy in SNs is often outweighed by the perceived utility [6], users are not willing to invest considerable effort into additional measures. Furthermore, many of the proposed solutions are still susceptible to unauthorised access by foreign governments. Therefore, we propose a novel and above all user-friendly approach to add confidentiality to social networks: Confidentiality as a Service (CaaS). By utilising existing infrastructure, this approach will allow us to conserve the existing user experience, scale well to the sizes of current social networks and overcome the legal problems many other approaches still have.

Our core contributions to confidentiality in social networks are:

- We demonstrate that many existing confidentiality-enhancing approaches for private SN messaging suffer from limitations, due to limited usability and legal issues.
- We introduce the CaaS paradigm that splits trust between a number of entities or organisations to overcome administrative and legal problems with confidentiality, often causing undesired access by foreign governments.
- We provide an easy-to-use prototype for Facebook that integrates seamlessly with the normal workflow and raises awareness for (un)protected information.

One may argue that encrypting the private messages of social network users may subvert the OSN provider’s business model. But, in contrast to other users’ content such as wall posts, Facebook explicitly mentions in their general business terms that users’ private messages are not analysed for targeted advertising purposes.

Organisation of this paper. In the next section, we will present the problems with privacy in modern online social networks and how other measures fail to address these. Section 3 will subsequently introduce the Confidentiality as a Service (CaaS) paradigm and how it integrates with social networks. In Section 4, we analyse the security of the CaaS approach before Section 5 gives details on the implementation and evaluation of a fully functional CaaS prototype for Facebook. Section 6 discusses related work, before Section 7 concludes this paper and outlines future work.

2. PROBLEM STATEMENT

In the following, we illustrate why privacy concerns are often neglected by users of Social Networks and outline how these needs could be satisfied. We will argue why existing paradigms, such as Public Key Infrastructures (PKI) and PGP (Pretty Good Privacy), are not easily compatible with many users’ habit of using the SN with multiple devices. Most importantly however, they are burdensome to use, mainly due to key-management issues. Also, there generally is no support for these technologies by social network providers, due to the reasons outlined in the previous paragraph.

Since today’s highly mobile Internet users access services from multiple devices and are not willing to invest additional effort into confidentiality measures, an effective solution for the message confidentiality problem must integrate well into this environment.

Based on the above problem statement, we derive the following usability requirements for an easy-to-use confidentiality and integrity mechanism for OSN messages.

Usability Requirements

UsabReq1. Confidentiality and integrity for OSN messages must be unobtrusively integrated into the users’ usage patterns, causing minimal extra effort. This includes the following sub-requirements:

\[\text{cf. } \text{http://www.facebook.com/terms.php}\]
• There should be no visible key-management for the user.
• Security mechanisms that are visible to the user should be limited to familiar concepts such as usernames and passwords.
• Encryption and decryption of data should be handled automatically by the software.

UsabReq2. Users must be able to recover access to their message history, in case decryption credentials were forgotten or lost.

UsabReq3. Users must be able to access encrypted data from multiple devices without needing to move digital artefacts.

2.1 Legal Problems

An entirely different risk to a user’s privacy stems from legal requirements. Many countries require telecommunication providers to allow national intelligence and law enforcement agencies access to their users’ communications. This prevents SN providers from offering complete privacy, since there is a requirement to be able to recover the cleartext of arbitrary users and there can be no guarantee that this recovery mechanism cannot be abused. A prominent example is Hushmail [17], who was forced to disclose personal emails (which the users believed to be fully confidential) from their web-based service after moving encryption from the client-side to the server-side. Because they were technically able to provide the plaintext (since they were then encrypting the emails on the server), they were legally required and forced to do so [24].

Hence, in order to ensure confidentiality for users, the storage of content and any corresponding confidentiality mechanisms should be managed by separated service providers, ideally subject to different legal systems. Involving multiple legal systems greatly improves the confidentiality of the users’ data. While data can be stored in a country with lower privacy standards such as the US, using a confidentiality provider residing in a legislation with strong privacy protection provides an effective means to avoid undesired disclosure.

3. CONFIDENTIALITY AS A SERVICE

The Confidentiality as a Service (CaaS) paradigm promotes the idea of outsourcing the provision of confidentiality for digital content. While traditional paradigms only involve the immediately participating parties (i.e., the communication partners in a private message exchange) into cryptographic operations, the CaaS paradigm proposes a third-party service provider who caters for encryption and decryption for the end users without itself being able to access the content being protected. The CaaS provider transparently provides key management and encryption/decryption of user content by linking a CaaS identity with the social network identity of that user. This link is used to effortlessly integrate confidentiality mechanisms into the traditional SN operations of the users, effectively freeing them from any security-related overhead. A CaaS provider can be run by a company, a non-profit organisation, a private person or any other entity that is interested in being a trust partner of the Social Network users. The CaaS provider is entirely oblivious to which kind of information is being protected. Its only task is to facilitate confidentiality and not to store, transport or access the users’ messages in any way. This is fully orthogonal to a SN provider’s interests and business model (cf. Req3 below).

We postulate that this separation of interests provides a powerful means to effectively protect data from undesired disclosure. Furthermore, this paradigm solves the legal problems described above, based on the sovereignty of countries and the ubiquitous nature of the Internet.

There are three requirements to successfully apply the CaaS paradigm:

Req1: The CaaS provider must be able to identify and authenticate SN users through an out-of-band channel, such as an Email address, to bind the SN identity to a CaaS identity.

Req2: The SN offers sharing of data between multiple users protected by an access control list mechanism.

Req3: The SN provider and the CaaS provider must not have any incentive to collude.

While Requirement 1 addresses the linking of service identities to prevent identity theft and will be discussed in Section 3.3.1, Requirements 2 and 3 outline the basic characteristics of a service that can leverage CaaS functionality.

3.1 The CaaS Workflow

Before delving into details of the CaaS paradigm, this section will outline a typical CaaS-enhanced message exchange, using Facebook as an example. In order to protect the privacy of a user’s message against a SN provider and the CaaS provider, multiple commutative layers of encryption are referred to as cLayer in the following – are applied. cLayers are layers of encryption that can be added and removed from data in an arbitrary order. For example, if two layers are added consecutively, either or can be removed first. Subsequently, we will use the following user-centric naming convention for the cLayers. Layers added locally by the SN user’s client are called local cLayers. Layers added by the CaaS provider on the remote CaaS server are called remote cLayers.

In a typical CaaS-encrypted communication session, one user – the sender – intends to confidentially send messages to one or more users – the recipients – over a SN platform. Figure 1 illustrates the CaaS paradigm and all involved parties.

The following describes the steps transparently and automatically executed in the CaaS workflow. First, the sender’s software adds a local cLayer to the message and sends it to the CaaS provider. Second, the CaaS provider adds a remote cLayer, that will remain in place until the recipients remove it. Before actually sending the encrypted message to all recipients over the SN platform, the sender’s software removes the local cLayer. Subsequently, the CaaS-cLayer-protected data can be transferred to the SN platform.

To recover the plaintext data, one recipient’s software again adds a local cLayer and sends the twice-encrypted data to the CaaS provider. At this point, the CaaS provider removes the remote cLayer and transfers the result back to the recipient. Finally, the recipient’s software removes the local cLayer to display the plaintext to the user. Since each user-to-CaaS-provider request is protected by a commutative en-
Studies have indicated that the use of a SN is not perceived as being particularly sensitive (e.g. online banking). Sures are often only accepted for transactions that are performed using web browsers and are familiar with identifying and authenticating using a tuple of username and password to access services online. The past has shown that further measures are often only accepted for transactions that are perceived as being particularly sensitive (e.g. online banking). Studies have indicated that the use of a SN is not perceived as a sensitive application (cf. [14, 27]) and users will therefore not accept complicated security measures. Hence, the CaaS paradigm maintains the familiar usability experience of a SN, in order to allow adoption by regular users. The integration of CaaS into the daily SN routine only requires one noticeable change, namely entering an additional CaaS password after logging into the SN and triggering the first encryption/decryption sequence. This password is sent directly and securely to the CaaS provider and cannot be intercepted by the SN provider.

3.3 CaaS Client-side Integration

Today’s social network users are used to viewing websites using web browsers and are familiar with identifying and authenticating using a tuple of username and password to access services online. The past has shown that further measures are often only accepted for transactions that are perceived as being particularly sensitive (e.g. online banking). As described by Garfinkel [13], email based identification and authentication (EBIA) is an attractive alternative to public-key infrastructure on the one hand and simultaneously provides functional security on the other. Internet users are used to this sort of workflow and therefore easily accept this measure. The EBIA scheme identifies and authenticates a user by sending a validation secret to the given email address. If the user is able to read the secret (i.e. to receive the email), the EBIA scheme was completed successfully and a new active CaaS credential set \[\text{Cred}_{\text{caas}} = (\text{email}, \text{ids} = [], \text{password})\] is created.

To provide the necessary integration with Social Networks, the CaaS approach binds an existing SN identity to a previously registered CaaS account (cf. Req1). We adopt a quick and simple out-of-band procedure to confirm the identity’s ownership before creating a binding. Most SN providers require an external identification token (in form of an email address) during their registration process. After the SN’s registration procedure, the internal SN identification (e.g. the Facebook UID) is bound to the user’s external identification. During registration for a CaaS account, this mapping can be leveraged to send a confirmation email to the future CaaS user using the EBIA scheme.

Using Facebook as an example, registration works as follows: The user logs into his CaaS account and clicks a button to log into his Facebook account using Facebook’s Authentication API. After Facebook has confirmed the authentication – through Facebook’s OAuth mechanism – the user agrees to allow the CaaS provider to see the email address he uses with Facebook. The CaaS provider uses this email address to send a validation link, which establishes that the currently logged-in CaaS user also has access to the Facebook account in question and can furthermore read email sent to that account. This process only proves that the current CaaS user has access to the SN account, but does not give the CaaS provider access to the SN account.

After the binding of a SN identity to a CaaS account was successful, the CaaS credential set is extended with the SN identity \[\text{Cred}_{\text{caas}} = (\text{email}, \text{ids} = [\text{sn极高}], \text{password})\]. When using Facebook, \(sn_{极高}\) is a large numerical UID. Subsequently, the user is able to use the CaaS provider’s services with all identities in \(\text{ids}\), since he has successfully proven that he is their owner.

The registration process has two requirements to ensure safe and correct functionality of the CaaS services:

**Req1** A user must choose different passwords for the CaaS and Social Network accounts – This prevents a
malicious SN provider from using the SN password to login to the CaaS service and vice versa.

**ReqReq2** The registration and binding procedure must be finished successfully before a user is able to participate in a CaaS-protected conversation. This prevents the pre-registration attack described in Section 4.1.

After the registration process has been completed, the user interaction to successfully encrypt and decrypt data with the CaaS is reduced to a minimum. The user only needs to install a web-browser plugin, which is a known point-and-click use case for many Internet users. The plugin transparently communicates with the CaaS provider and handles all the cryptographic operations without requiring any more user involvement, except entering the CaaS password once per session. The duration of a session is configurable and can be based on a SN login or a timeframe.

After a successful registration, the account binding, and the installation of the plugin, the next step is to actually encrypt and decrypt Social Network messages. The next section will outline the algorithms used for the actual data encryption and decryption on the client- as well as the server-side.

### 3.3.2 +cLayerLocal

The +cLayerLocal procedure is applied each time before a message is sent to the CaaS provider from the client-side. As input parameters, it takes the cleartext message $\text{clear}_{u_1}$ as well as the list of recipients $R$ and subsequently runs the following steps without any user involvement:

1. Choose a random initialisation vector $\text{iv}_{u_1}$ and temporarily store it.
2. Choose a random symmetric encryption key $k_{u_1}$ and temporarily store it.
3. Calculate the message digest $\text{dig}_{u_1} = H_u(\text{clear}_{u_1})$. $H_u$ is a public parameter published by the corresponding CaaS provider.
4. Calculate a key stream $k\text{str}_{u_1} = \text{Sym}^{\text{stream}}(\text{iv}_{u_1}, k_{u_1})$. $\text{Sym}^{\text{stream}}$ is a stream cipher chosen by the client software.
5. Add a first cLayer to the plain text message: $\text{enc}_{u_1} = \text{clear}_{u_1} \oplus k\text{str}_{u_1}$.

The result of the +cLayerLocal procedure is a first cLayer of encryption on the original message. This ciphertext is sent to the CaaS provider for the next step in the process.

### 3.3.3 -cLayerRemote

The +cLayerRemote procedure adds the remote cLayer to the message and is applied after a CaaS user successfully authenticated with his CaaS credentials. Because of the previous binding, the social network user ID in combination with the CaaS password is sufficient. The input for the +cLayerRemote procedure are the sender’s credentials for authentication, the ciphertext $\text{enc}_{u_1}$ that was created by +cLayerLocal and the list of recipients $R$ (which is automatically extracted from the SN). In case the authentication of the sender was successful, the CaaS provider software runs the following steps to add a remote cLayer to the ciphertext $\text{enc}_{u_1}$:

1. Check if all recipients in $R$ are registered CaaS users. Abort if not (cf. ReqReq2).
2. Add the sender’s identity to $R$ and sort the list to obtain $R'$.
3. For $r_j \in R'$ run $H_p(r_j) = H_p(H_p(r_{j-1}))$ with $j \in \{1, \ldots, i = |R'|\}$ to obtain an iterative hash $h_i$ of all participating users.
4. Calculate the symmetric secret encryption key $k_p = HKDF^p(h_i, X_p)$.
5. Choose a random initialisation vector $\text{iv}_p$.
6. Calculate $k\text{str}_p = \text{Sym}^{\text{stream}}(\text{iv}_p, k_p)$, the key stream used to encrypt the message.
7. Add a remote cLayer to the sender’s input: $\text{enc}_p = \text{enc}_{u_1} \oplus k\text{str}_p$.
8. Send the tuple $(\text{iv}_p, \text{enc}_p)$ back to the requesting sender over a secure communication channel.

### 3.3.4 -cLayerLocal

The -cLayerLocal procedure is used to remove the initial local cLayer, added during +cLayerLocal, after a message is returned from the CaaS provider. It takes the CaaS provider’s result $(\text{iv}_p, \text{enc}_p)$ and $\text{iv}_{u_1}$, $k\text{str}_{u_1}$ from +cLayerLocal as inputs. It calculates $\text{rts}$, the ciphertext ready to be sent to the SN provider, as follows:

1. Decrypt $\text{enc}_p$: $\text{rts} = \text{enc}_p \oplus k\text{str}_p$.
2. The message $(\text{rts}, \text{iv}_p, \text{dig}_{u_1})$ is ready to be sent to all recipients $R$.

**Figure 2** summarises the sequence to CaaS-encrypt Social Network messages. To recover CaaS-encrypted messages, a recipient’s client software has to follow three steps:

First, a cLayer is added to the received message by running the +cLayerLocal algorithm (see above) that sends the result to the CaaS provider. On the CaaS provider’s side, the -cLayerRemote algorithm is executed and removes the CaaS’s remote cLayer from the data before another call of
-cLayerLocal removes the final local layer of encryption. Figure 3 illustrates the sequence of operations that have to be executed after a message was received to decrypt a CaaS encrypted message.

3.3.5 -cLayerRemote

The -cLayerRemote procedure is executed after a CaaS user successfully authenticated. Its inputs are the credentials of a recipient \( r \in R \) for authentication, the ciphertext \( enc_r \) received by \( r \), the initialisation vector \( iv_p \), and the list of participating users \( R \) including the sender. In case the authentication of \( r \) was successful, the CaaS provider runs the following steps to remove its cLayer from \( enc_r \):

1. Sort the list \( R \) to obtain \( R' \).
2. For all \( r \in R' \) run \( H_p(r_j) = H_p(H_p(r_j-1)) \) for \( j \in 1, \ldots, |R'| \) to obtain an iterative hash \( h_i \) of all participating users.
3. Calculate the symmetric secret encryption key \( k_p = HKDF(h_i, X_p) \).
4. Calculate the decryption key stream \( kstr_p = Sym_{\text{stream}}(iv_p, k_p) \).
5. Decrypt \( enc_r \): \( dec_p = enc_r \oplus kstr_p \).
6. Send \( dec_p \) back to the requesting client over a secure communication channel.

On reception of the CaaS provider’s response, the message is still protected by the recipient’s local cLayer. To remove this last protection layer, the recipient runs the -cLayerLocal procedure on \( dec_p \). Now the recipient holds the cleartext \( clear_r \) and can validate the message integrity using the message digest \( digest_{clear} \).

In case no adversary manipulated the data, \( clear_r \) is the same as \( clear_{u_1} \). For clarity, a short summary of the multiple encryption/decryption steps is given:

1. \( u_1 \) adds a cLayer: \( enc_{u_1} = clear_{u_1} \oplus kstr_{u_1} \).
2. The provider adds a cLayer: \( enc_p = enc_{u_1} \oplus kstr_p \).
3. \( u_1 \) removes his cLayer: \( rts = enc_p \oplus kstr_{u_1} \).
4. \( rts \) is sent to (and stored at) the SN provider and received by \( r \).
5. \( r \) adds a cLayer: \( enc_r = rts \oplus kstr_r \).
6. The provider removes its cLayer: \( dec_p = enc_r \oplus kstr_p \).
7. \( r \) removes his cLayer: \( clear_r = dec_r \oplus kstr_r \).

\( clear_{u_1} \) can hence be recovered from \( dec_p \) because XOR is commutative and self-inverse:

\[
\begin{align*}
dec_p &= (((((clear_{u_1} \oplus kstr_{u_1}) \oplus kstr_p) \oplus kstr_r)) \oplus kstr_r) \\
&= clear_{u_1} \oplus (kstr_{u_1} \oplus kstr_u) \oplus (kstr_p \oplus kstr_r) \\
&= clear_{u_1} \oplus 0 \oplus 0 \oplus 0 \\
&= clear_{u_1}
\end{align*}
\]

With a truly random key that is used only once, XOR encryption is called a One-Time Pad [23]. There are different symmetric encryption schemes, such as AES [20], that can be used in stream cipher mode, generating a key stream of arbitrary length. While a One-Time Pad encryption scheme is provably secure, the security of a stream cipher depends on the randomness characteristics of the Pseudo-Random Number Generator used to generate the key stream (cf. [19]).

4. SECURITY ANALYSIS

There are several possible attack vectors in the Confidentiality as a Service paradigm. These can be divided into internal and external attacks. An internal attack can be launched by any of the involved providers. For example, a malicious Social Network provider could try to decrypt data in the name of a CaaS user (mounting a Social Network impersonation attack). A second possible internal attack could be driven by a malicious CaaS provider that tries to request data from the Social Network provider in the name of a Social Network user (mounting a Social Network identity impersonation attack). An external attack could be mounted by a Social Network user that tries to illegally encrypt or decrypt a message and was originally not involved in the communication session. Subsequently these attacks are briefly discussed.

4.1 Malicious SN provider

The Social Network provider is responsible for storing the messages for its users and granting access to the messages to authorised users. To encrypt or decrypt a CaaS encrypted message in the name of a SN user, the SN provider would need to have access to the user’s CaaS credentials. Since we require in RegReq1 that the CaaS password must be different from the Social Network password, the SN provider is not able to encrypt or decrypt messages using the CaaS service in the name of a user by abusing password information.

In a second possible attack, a Social Network provider would register a CaaS account for one of its users. Upon detecting an encrypted message, the SN provider tries to impersonate one of the users involved in that conversation and register for a CaaS account, before that user can herself register with the CaaS provider. Therefore, we call this a pre-registration attack. As a result, the SN provider would...
be able to decrypt the message. Because the SN provider is responsible for authentication on its platform, it can make the CaaS provider believe that he is indeed authorised to access this user’s account by faking the authentication response. In order to prevent this kind of attack, we implement the following countermeasure to minimise the threats of such a pre-registration attack.

When Alice intends to send a CaaS-encrypted message to Bob, the +LayerCaaS algorithm (cf. Section 3.3.3) verifies the existence of a CaaS binding for Bob. If no such binding exists, the CaaS provider will deny the encryption. Thus, encrypted messages can only be sent, if the receiving SN users already have a CaaS account bound to their SN identities. Hence, the SN provider cannot register CaaS accounts for encrypted messages on-demand. The only remaining option for the SN provider is to preemptively pre-register CaaS accounts for all of its users (which can be easily detected, since none of them would be able to create a valid CaaS account). We believe that this measure has very little usability impact compared to the security benefit.

4.2 CaaS Provider

Messages presented to the CaaS provider are always protected by a Layer added by a CaaS user. During encryption, message \( m_1 \) is encrypted with the user’s keystream \( kstr_{u_1} \). Hence, the CaaS provider only sees \( enc_{u_1} = cleartext \oplus kstr_{u_1} \) and never gets in contact with \( cleartext \). To decrypt the received message \( rts \), a CaaS user \( u_2 \) adds another Layer \( enc_{u_2} = rts \oplus kstr_{u_2} \) and sends \( enc_{u_2} \) to the CaaS provider. In both situations, a malicious CaaS provider does not have enough information to obtain the message’s cleartext because one commutative Layer protects the message from being readable by the CaaS provider.

4.3 External User

An external user is one that cannot legitimately access the conversation, but may be in possession of a valid SN and CaaS account. First, as long as the SN access control mechanisms are working, this user should even not be able to access the encrypted data. Second, this user is also not capable of requesting decryption for the data at the CaaS provider, since his CaaS account is not part of the key for the data. Hence, an external user needs to get access to the ciphertext (i.e. break into the SN provider or steal a participant’s SN account) and the decryption service (i.e. break into the CaaS provider or steal that participant’s CaaS account).

5. IMPLEMENTATION

We provide both, a prototypical CaaS server implementation as well as a Greasemonkey user script, which integrates our CaaS approach into Facebook. We utilise the Advanced Encryption Standard (AES) in Counter Mode as stream cipher. AES is an industry standard symmetric cipher, that provides multiple stream cipher modes such as cipher feedback (CFB), output feedback (OFB) and counter mode (CTR). While all provide excellent security characteristics, CTR has significant efficiency advantages without weakening the security of the generated stream cipher (cf. [22]). In order to prevent key reuse attacks (cf. [10]), each encryption-decryption sequence includes the selection of a new random initialisation vector. XOR-based encryption additionally requires an out of band mechanism to check message integrity, as used by the +LayerUser and +LayerUser algorithms. For that purpose, SHA-256 as a cryptographically secure hash function was selected.

We assume that most Facebook messages are short and typically not longer than about 8000 characters, which is more than three A4 pages of text without breaks. Consequently, we directly encrypt the messages. If larger pieces of data need to be protected, a bootstrapping scheme can be applied. In such a scheme, the client-side plugin only CaaS-encrypts a session key which is then prefixed to the message encrypted with that session key.

The server-side implementation is based on a Java REST webservice, using a relational database to store user accounts. Our webservice implementation can be run in any Java application container and hence is easily deployable without much effort. To ensure trusted communication between a client and the CaaS provider, TLS with a server-side certificate is used. The RESTful webservice is a comfortable interface for programmers to access the CaaS service API and simplifies the development of additional client applications.

As argued above, client-side usability is of utmost importance in order to gain the users’ acceptance for privacy measures. Our intention was to implement a user registration interface that today’s Internet users are accustomed to. Therefore, we provide an easy-to-use website to register for a new CaaS account and to bind existing Social Network accounts to. The CaaS account registration requires only an email address and a password. This credential set was chosen since these identification tokens are most frequently used and offer a good usability-to-security-ratio for this scenario. To bind a Facebook account to a CaaS account, the website utilises Facebook’s Authentication API [6]. The user authenticates to Facebook and in return provides the CaaS provider with proof of authentication, his Facebook UID as well as the registered email address, as described in Section 3.3.1. After fetching both properties from Facebook – given the user’s approval – these values are bound to the corresponding CaaS account.

At this point the binding is still inactive and requires the confirmation of the Email address given by Facebook. For that purpose, a confirmation link is sent to the email address which the user has to click on, in order to validate the binding. This is a standard procedure and proved to be easily accepted during our user study (cf. below).

After a successful binding of the Facebook identity, we use a Greasemonkey script to be able to encrypt and decrypt messages within the regular Facebook UI. Greasemonkey is a Mozilla Firefox extension that allows end users to install third party JavaScript extensions, that can manipulate a displayed website.

The CaaS client currently supports the sending and receiving of CaaS encrypted messages through the regular Facebook user interface without requiring any understanding of cryptographic artefacts at the end-user’s side. Figure 4 shows the modified message composer. When the user clicks the "New Message" button on the Facebook website, the CaaS script hooks into the message form. There is one listener that checks for new recipients and verifies if they are available for encrypted messaging (cf. Section 1.1). A
second listener hooks into the "Send" button routine to run the required CaaS encryption sequence before sending the message through the Facebook messaging system to all recipients.

Figure 4: The modified Facebook message composer.

If a recipient was selected that already has an active Facebook-to-CaaS binding, its label is highlighted in green. In case the selected recipient does not have an active binding, its label is highlighted in red and a warning is displayed, that says that the message will be sent in clear and not CaaS-encrypted. This mechanism gives unambiguous feedback to the user and explains the current status of the message’s security. If all recipients are valid recipients and the sender hits the "Send" button, the CaaS encryption sequence is triggered. In case no CaaS client session is active, a popup window is displayed that asks for the user’s CaaS password. This password is cached for a configurable amount of time for the user’s convenience. The added markers on the message box and the send button provide a concise visual indicator that this message will be protected.

Similar to sending a CaaS encrypted Facebook message, the script supports on-the-fly decryption and reading of cleartext messages. Therefore, the Facebook messages site is modified and parsed for CaaS encrypted messages. If such a message is found, the ciphertext is extracted and the CaaS decryption sequence transparently and automatically executed (cf. Section 3.1). On success, the client-side script locally removes the last cLayer and replaces the ciphertext with the recovered cleartext message in the site’s DOM tree. Greasemonkey’s sandbox environment prevents Facebook from inserting their own hooks into the DOM tree to eavesdrop on the messages after decryption. Figure 5 shows a comparison of the Facebook message site with and without CaaS-decryption.

If a CaaS-encrypted message was detected and successfully decrypted, a green border indicates that this message was protected using CaaS-encryption. Again, if no CaaS password was cached, a popup is displayed and asks the user to enter his password to decrypt the found messages.

To help users to intuitively understand that message protection is in place, protected content is annotated with visual indicators. Several studies (e.g. [29, 28]) found that visual privacy indicators outside of the actual website, such as little locks indicating a valid TLS session, are often overseen or ignored in the presence of a correct-looking web page content. Egelman et al. also found that the timing and placement of privacy indicators plays a central role in the user’s perception [10]. Users were ready to pay a higher price for a product while online-shopping, if a privacy indicator showed that a shop provided better privacy. However, this indicator only worked when placed immediately next to the offer.

Thus, in our prototype, a red border marks a piece of information that is potentially in need of protection or is unprotected, while a green border intuitively indicates successful protection. We believe that the visual indicators will also raise the user’s awareness for private information in need of protection, and will therefore eventually increase the user’s privacy perception and actual privacy.

Since our client-side prototype is written in Javascript, it is easily verifiable that the CaaS-client itself works as intended and does neither send any clear-text to the CaaS service nor does leak any other privacy related information.

Evaluation.

We conducted a user study with 20 undergrad students and found that registering for the CaaS service and binding a Facebook account took 3:08 minutes on average (ranging from 90 seconds to 6 minutes, all users could successfully finish the task). Additionally, we collected performance measurements to estimate the delay a user experiences when using CaaS-enhanced social networks. Our results indicate that there is only very little additional delay introduced to the regular Facebook webpage. In our experiments, it took between 5 to 6 seconds for the Facebook page to finish all relevant AJAX calls on the initial visit and about 1.5 seconds for subsequent interactions. On average, it took between 33ms and 154ms to CaaS-encrypt or -decrypt a message of 2 to 8000 characters (cf. Fig. 6), including network-based delays of a broadband cable provider (about 25ms round-trip time for an ICMP ping packet) and TLS session establishment. The tests were run on a 2.66 GHz Core2 Duo machine with 8 GB of RAM against a 3 GHz Pentium D dual core server having 4GB of RAM.

Using the proposed mechanism, encrypting a message upon
sending or decrypting upon reception is only barely noticeable by the user. Asynchronously decrypting the entire message history (typically the past 30 messages) takes on average as long as it takes to load the page (between 222ms and 4101ms depending on the message size) and can begin while the rest of the page is still loading. We therefore believe that our confidentiality plugin does not disturb the normal Facebook experience.

A prototypical CaaS provider instance and the Facebook Greasemonkey plugin are available at https://cloudcrypt.to to practically evaluate the service and verify the results.

6. RELATED WORK

There are several recent approaches to secure content in social networks, especially Facebook. In 2008, Lucas et al. [15] proposed flyByNight, a prototype Facebook app that encrypts and decrypts messages using public key cryptography. The flyByNight server handles the key management and uses its own database to store the encrypted messages. By storing all encrypted messages in their database, flyByNight’s resources need to scale as Facebook does, which can be very expensive. Additionally, by using Facebook’s App API, all requests and all responses of the application travel through Facebook’s servers, whereas in the CaaS approach, Facebook is not aware of any operations beyond regular message sending. Lucas et al. hinted at the necessity of usability, but did not show any usability features beyond stating that a privacy extension would need to be web-based, have good performance and require little technical knowledge. However, flyByNight requires the user to leave the regular Facebook UI experience and use another app for their messaging instead, which would lack all the convenience and usability features that Facebook provides.

Scramble! [4] is a PKI based Firefox plugin that stores encrypted social network content at a TinyLink server or stores it directly at the SN provider. However, key management is again complicated and relies on PGP mechanisms and must be dealt with explicitly by the user. Most notably, you need to be carrying your private key around in order to make this approach work on multiple devices.

Another approach was taken by Guha et al. [15], who use shared dictionaries to map different “atoms” of information to a similar, valid piece of information. For example, Alice’s address would be randomly replaced by Bob’s, according to some mapping key. Their NOYB prototype can hide the fact that content is being protected but also necessitates key exchange using email. Additionally, reusing other users’ information can have privacy implications of itself.

Baden et al. [5] present Persona, a privacy enhanced social network using public key cryptography and attribute-based encryption (ABE). They acknowledge the need to integrate their service with the popular networks and demonstrate a prototype that provides their services as a Facebook application. They argue that existing SN apps can be gradually migrated to use the Persona platform, at least for storage. Using the Facebook API, it is however not possible to access the messaging service, which might be considered to be one of the features with the most utility. The Persona platform also comes with the key management issues discussed before.

uprotect.it [21] also offers confidentiality services for Facebook. However, their service comes with several privacy problems. Because uprotect.it not only provides confidentiality but also stores the user content on their servers alongside the encryption keys, they are able to eavesdrop on the users’ data, as stated in their Terms of Service. encipher.it [11] provides a very lightweight approach, using a simple bookmarklet to apply symmetric AES encryption to arbitrary text. Sharing the key with whoever is supposed to decipher a message is entirely left to the user. Naturally, this approach does not integrate well with the normal user-experience of sending Facebook messages. There also is a possibility to use a Firefox GPG plugin with Facebook, so that inline ciphertext is automatically decrypted. However, a valid public and private key pair is necessary and further key management issues are not addressed by this approach.

Anderson et al. [2] presented a concept to use rich-clients as a way to improve privacy. The SN provider is reduced to a mere content distribution server while the client handles cryptography and information semantics. Again, this approach would require a user to migrate to another SN and change the interaction patterns. In a similar fashion, a number of projects (e.g. [7, 9, 26]) proposed to distribute a Social Network across a peer-to-peer infrastructure, thereby removing the risks inherent to central service providers. However, these approaches struggle to gain broad acceptance and are often only used by privacy-aware or expert users. One of the central ideas of the CaaS approach is to protect existing and proprietary services transparently and therefore protect a large number of users that are already experiencing privacy problems.

None of the above approaches achieve the ease-of-use and non-intrusive integration into existing SN services offered by the CaaS paradigm.

7. CONCLUSION

In this paper, we presented a new paradigm for user-friendly, non-obtrusive confidential messaging for social networks. The key features of the paradigm are the separation of concerns, which protects the user from malicious SN and CaaS providers, and the usability concept, which eliminates all key-management issues for the user. We leverage the messaging workflow and the intuitive UI of existing social networks, which allows users to gain an intuitive awareness of their confidential messaging. We have also presented a rapid prototipo
of privacy as well as an ability to directly see which content is protected and which isn’t. Since the CaaS provider does not need to store any keys or data, our paradigm has a very small resource footprint and hence has the potential to scale more easily to the sizes required by modern SN platforms. A user study showed that the CaaS enrolment procedure is quick and easy and requires no specialised knowledge. Our Facebook plugin demonstrates that privacy mechanisms can be transparently integrated into existing user interfaces with-out interruption of the user’s regular workflow.

In future work, we will examine how multiple CaaS providers can be chained to increase the level of security. Furthermore, CaaS integration efforts for multiple other services, such as Twitter, Email and Dropbox, are underway.

8. REFERENCES


