A

Virtual Private Social Networks and a Facebook Implementation

MAURO CONTI, University of Padua
ARBNOR HASANI, Vrije Universiteit Amsterdam
BRUNO CRISPO, University of Trento

The popularity of Social Networking Sites (SNS) is growing rapidly, with the largest sites serving hundreds of millions of users and their private information. The privacy settings of these SNSs do not allow the user to avoid sharing some information (e.g. name and profile picture) with all the other users. Also, no matter the privacy settings, this information is always shared with the SNS (that could sell this information or be hacked). To mitigate these threats, we recently introduced the concept of Virtual Private Social Networks (VPSNs).

In this work we propose the first complete architecture and implementation of VPSNs for Facebook. In particular, we address an important problem left unexplored in our previous research—that is the automatic propagation of updated profiles to all the members of the same VPSN. Furthermore, we made an in-depth study on performance and implemented several optimization to reduce the impact of VPSN on user experience.

The proposed solution is lightweight, completely distributed, does not depend on the collaboration from Facebook, does not have a central point of failure, it offers (with some limitations) the same functionality as Facebook, and apart from some simple settings, the solution is almost transparent to the user. Thorough experiments, with an extended set of parameters, we have confirmed the feasibility of the proposal and have shown a very limited time-overhead experienced by the user while browsing Facebook pages.

Categories and Subject Descriptors: K.4 [**Privacy**]: Abuse and crime involving computers; K.6.5 [**Security**]: Protection

General Terms: Security

Additional Key Words and Phrases: Virtual Private Social Networks, Internet and Privacy, Social Networking Sites, Facebook Privacy

ACM Reference Format:
Conti, M., Hasani, A., and Crispo, B., 2012. Virtual Private Social Networks and a Facebook Implementation. ACM Transactions on the Web V, N, Article A (January YYYY), 28 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

This paper is an extended version of the paper "Virtual Private Social Networks", appeared in the Proceedings of the First ACM SIGSAC Conference on Data and Application Security and Privacy (CODASPY 2011), by the same authors.
This work was partly supported by: (1) the Dutch STW-Sentinel S-MOBILE Project, contract no. VIT.7627; (2) the European Project Network of Excellence NESSoS, contract no. FP7-256980; (3) the TENACE PRIN Project (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research; (4) Mauro Conti is supported by a Marie Curie Fellowship funded by the European Commission for the PRISM-CODE project (Privacy and Security for Mobile Cooperative Devices) under the agreement n. PCIG11-GA-2012-321980.
Contact author: M. Conti - conti@math.unipd.it.
Authors' addresses: M. Conti, Department of Mathematics, University of Padua, via Trieste 63, 35131 Padua, Italy; A. Hasani, Computer Science Department, Vrije Universiteit Amsterdam, De Boelelaan 1081a - 1081 HV Amsterdam, The Netherlands; B. Crispo, Computer Science Department, University of Trento, via Sommarive 14, I-38123 Povo (TN), Italy.
The authors thank Frank Mennink for his help in implementing a part of FaceVPSN and for running some of the experiments.
The aim of this work is only to present a possible approach to protect users' privacy and show its technical feasibility—by no means we invite users not to follow Facebook terms of services, and we warn the users that (at the time of writing) adopting the technique presented in this work might result in consequences which include their Facebook account being deactivated.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Social Networking Sites (SNSs) are Internet-based applications that allow for user-generated content to be published and accessed easily by a global audience. Some of the SNSs are the most visited sites on the Internet. The biggest SNS, i.e. Facebook (which number of users [Facebook 2012a] surpassed the population of the European Union), allows users to create an account with an extensive choice of profile information fields (e.g. Basic Information, Interests, Profile Picture, Contact Information, etc.).

The pervasiveness of social networks, in a growing number of people's life and activities, has led to increasingly more private information being "leaked" online. In many cases (i) this is due to just carelessness of users. In many other cases, (ii) information is "leaked" because of the design of privacy controls of the SNSs. As an example, in Facebook, Name, Profile Picture, Gender, and Networks, are considered "publicly available information" (before a recent change [Facebook 2012a], even Current City, Friend List and Pages one is a fan of were considered public). That is, such information, according to Facebook may *"be associated with you (i.e., your name, profile pictures, cover photos, timeline, User ID, username, etc.) even off Facebook, can show up when someone does a search on Facebook or on a public search engine, will be accessible to the Facebook-integrated games, applications, and websites you and your friends use, and will be accessible to anyone who uses our APIs such as our Graph API"* [Facebook 2012b]. Furthermore, not only can anyone view these fields by simply knowing someone's user ID or username, but also they can be accessed by third party application developers and websites. In other words, a Facebook user cannot abstain from sharing certain information with other third parties. Finally, (iii) inference (of private attributes) can be accomplished by joining data of the profiles the user has in different SNSs (e.g. Facebook, LinkedIn, Twitter, Flickr)—while harder to get, information available from the real world could also be used. Consequently, the privacy related issues have raised the interests of security researchers.

The significance of the privacy issue is escalated by the fact that most of the profiles contain real information [Gross and Acquisti 2005; Young and Quan-Haase 2009]. According to [Young and Quan-Haase 2009], in Facebook 99.35% of the users use their real name, 92.2% reveal their birth date, 80.5% their current city, and 98.7% also display a profile picture of themselves. Furthermore, it is relatively easy to re-identify a user that has multiple (public) accounts in various SNSs [Narayanan and Shmatikov 2009].

There are countless reports on how this amount of data collected from SNSs are being (mis)used. For instance: marketing companies are getting community structure data [Dybwad 2010]; identity theft [Tabakoff 2009] and identity cloning [Jin et al. 2011] can be performed more easily; students or employees are getting in trouble because of information that was found on social networks through de-anonymization techniques [Wolfe-Wylie 2010]; and even the robbers' life has been simplified [van Amstel et al. 2010]! Furthermore, a recent research [Reay et al. 2009] shows that web sites (including SNSs) generally do not even claim to follow all the privacy-protection mandates in their legal jurisdiction (this is true for several jurisdiction with specific laws for privacy, including several European Union nations). All these reasons explain the fact that some people do not like that anyone (including SNSs) knows their personal information as: real name, city, gender, interests (e.g. via Facebook "networks"), or their social network graph (the people to whom they are connected), and other information that might be available, depending on the specific SNS.

This work addresses these privacy issues by leveraging the concept of Virtual Private Social Network [Conti et al. 2011]; a concept similar to the one of Virtual Private Network (VPN), used in computer networks. The main idea is to build private social networks between users leveraging the already existing architecture of SNSs. It is essential for a VPN to benefit from the existing infrastructure. Members of the VPN are network nodes, and they can communicate in a way that is confidential with regards to the nodes outside that VPN. Analogously, a VPSN leverages the infrastructure of an already existing social network, primarily because it is very unlikely for users to move away from popular social networks. Members of the VPSN are already members of the social network they leverage. VPSN members can share information (specifically, the attributes of

the profile) in a way that is confidential with regards to: i) the other members of the underlying social network; and ii) also against the social network administrator. A VPSN can be seen as: i) a privacy-aware SNS without the need to manage a SNS infrastructure; ii) a tool to mitigate the privacy threats for the huge number of users that are already using popular SNSs.

We emphasize that the concept of VPSN is not present yet in current SNSs. That is, a user is not able to implement a VPSN with the available tools. While it is not clear whether privacy is a motivation strong enough to justify the implementation of a new SNS from the scratch, we believe the first SNS platform able to offer strong privacy will have a major competitive advantage [Cutillo et al. 2009]. Our focus is on leveraging existing infrastructure, and mitigating the privacy problem for SNSs that already count millions of users.

We focus more specifically on Facebook—the most popular SNS—for which we implement our concept of Virtual Private Social Networks (VPSNs). We call this implementation FaceVPSN. The central problem being addressed is how to use Facebook to share real information only with the other VPSN's member—without any other third party (including application developers) being able to see the real information.

The robustness of FaceVPSN lies in the fact that it is technically compatible with Facebook and it does not require its collaboration, while it completely leverages the already present and (freely) available Facebook architecture and systems—without requiring any additional infrastructure as other solutions do (e.g. FaceCloak [Luo et al. 2009] and NOYB [Guha et al. 2008]). In FaceVPSN, the required system resources are distributed among the VPSN members: each member is required to have a small storage and computation overhead that are proportional to the number of members in the VPSN the user belongs to.

Contribution. In this paper, we introduce the concept of Virtual Private Social Networks (VPSNs) and we propose its first complete architecture and implementation for Facebook: FaceVPSN. The proposed solution is lightweight, completely distributed, does not depend on the collaboration from Facebook, does not need to rely on any specific server (i.e. it is also resilient to failure or malicious activities from external XMPP [XMPP Protocol 2011] servers involved in our architecture), it offers almost the same functionality as Facebook, and it is (to some extent) transparent to the user. Thorough experiments confirmed the feasibility of the proposal, and have shown a very limited time-overhead experienced by the user while browsing Facebook pages.

While VPSN recalls to some extent the concept of VPN, addressing privacy in social networks sites is not as trivial as adding encryption to the communications (as it would be for a VPN). Rather, the analogy relies on the fact that VPSN is enabling two parties to convey information to each other by hiding it to the infrastructure that the parties use to communicate. As an example, encrypting profile information and messages in Facebook may result in having your profile removed by the SNS. Indeed, a weakness that is common among most of the current privacy preserving solutions for Facebook is the following: Facebook is able to identify these solutions, then either it can remove the solutions from the social network, or remove the users that use those solutions. Differently from those approaches, our solution is far more stealthy. In particular, we argue that for Facebook it is not easy to identify a profile with a reasonable name but which is different from the real name of the owner; also, from recent version of browsers (e.g. Firefox 3.0) it is not possible for Facebook also to identify whether the user is running FaceVPSN add-on in her browser [Mozilla 2012a].

We clarify that our aim is not to make yet another new privacy aware social network, as there are already in the literature [Daniel et al. 2010], but to mitigate the privacy threats of a number of users already using Facebook—please note that this number is almost as the population of the entire Europe and USA combined.

Organization. The rest of the paper is organized as follows. Section 2 introduces related work in the field of privacy issues in social networks. Section 3 introduces the new concept of Virtual Private Social Networks (VPSNs). Section 4 introduce FaceVPSN, our implementation of VPSNs for Facebook. Design and implementation details are discussed in sections 5 and 6, respectively. Section 7 contains the evaluation and discussions about the proposed solution. Finally, in Section 8 we present our concluding remarks.

2. RELATED WORK

Following the popularity of social networks, the issue of privacy in social networks has gained momentum in the last couple of years. Nonetheless, most of the published research deals primarily with general de-anonymization, and in few cases with anonymization techniques that are not designated for a specific SNS. We shall also bring to the reader's attention the fact that most of these works, referring to Facebook, have been made obsolete by the recent changes (e.g. in September 2011) of the Privacy Policy of Facebook [Facebook 2012b].

Many papers investigate the different facets of privacy in SNSs. The authors in [Mislove et al. 2010], [Zheleva and Getoor 2009], and [Hay et al. 2007] emphasize the issue of communities centered around attributes (i.e. Facebook Fans of Vrije Universiteit in Amsterdam) when approaching privacy problems. They show the possibility to predict, with high probability, a user's attributes using the attributes of her friends and contacts. In [Golbeck 2009], the authors investigate on profiles similarity and how those relate to the way users determine trust, hence their willingness to share private information. [Kacimi et al. 2009] presents a privacy protocol allowing anonymous opinion exchanges over untrusted SNSs. In [Korolova et al. 2008], the authors investigate the number of user accounts needed to be compromised to rebuild the entire social graph. This number is strictly dependent on the SNS's lookahead (a social network has lookahead l if a user can see all of the edges incident to the nodes within distance l from her). The paper recommends to limit the lookahead to 1 or 2, for any social network that wishes to decrease its vulnerability.

Researchers already proposed several solutions to protect private data in social networks. In [Aimeur et al. 2009] the authors proposed a user privacy policy (in the form of a contract) to ensure the proper protection of private data. SNSs are supposed to implement such policies to improve the privacy of their users. Despite a prototype implementation being presented in [Aimeur et al. 2010], to have an impact the solution requires existing SNSs to adopt such a policy. Similarly, Persona [Baden et al. 2009] is a social network in which the user defines a privacy policy. This policy manages access to user information. Persona uses an application named Storage to allow users to store personal data, as well as share them through an API with others. The authors have integrated Persona with Facebook as an application to which users log-in through a Firefox extension, which interprets the special markup language used by Persona applications. Only users of Persona, with necessary keys and access rights, will be able to access the data from Storage, and view them in their browser. Nonetheless, it is only another Facebook application that can easily be removed by Facebook from the applications directory. Another interesting recent thread of research is considering the possibility of new competitor social networks designed with privacy in mind [Cutillo et al. 2009; Vu et al. 2009; Durr et al. 2010; Daniel et al. 2010; PrimeLife 2011]. We expect those solutions, when supported by a good business model, to reach a wide consensus from users. However, differently from these solutions, our aim is to mitigate threats to the privacy of millions of users that are currently using existing SNSs. That is, we aim at leveraging (for free) the existing SNSs infrastructure to build on top of them Virtual Private Social Networks (VPSNs).

In [Felt and Evans 2008], the authors propose the "privacy by proxy" API to allow Facebook to reveal users' private information only if this is explicitly allowed by the user. This is done by interpreting special FBML (Facebook Markup Language) tags (i.e. Facebook specific HTML-like tags) in the output of the applications. However, the solution in [Felt and Evans 2008] requires the cooperation of Facebook. Similarly, another proposal, flyByNight application [Lucas and Borisov 2008], requires the cooperation of Facebook. flyByNight provides secure messaging between Facebook friends, by encrypting and decrypting sensitive data using client-side JavaScript. However, it does not solve the privacy concerns with regards to publicly available information. Another solution for privacy of communication in SNSs is the Secret Interest Groups (SIGs) framework [Sorniotti and Molva 2010]. Although, not a direct solution to privacy of public information, it offers an interesting new way to create self managed "secret groups" (about private topics), and control group memberships. The idea is integrated with Facebook as an implemented Java HTTP proxy. While the concept of SIGs is close to the one of VPSNs, we note that the solution suggested in [Sorniotti

and Molva 2010]—even if not centralized or having a single authorization entity—envisages the presence of one or more managers that, for example, allow new users to join the SIG. Furthermore, even for the managers, any operation is "thresholdized". That is, more managers need to join and agree in order to make a decision (e.g. add a new member). We underline that such an approach is completely different from the one we propose in our work. That is, each single user is completely independent on defining who (and how) will be able to access her private information. Finally, for all the solutions implemented through a proxy, we argue that a user might not be willing to have a proxy filtering all of her browsing.

Another similar idea of grouping friends is implemented as a Facebook application in [Yuksel et al. 2010]. In this solution, some questions are posed to the user (e.g about her willingness to share a piece of their personal information with a friend). Based on her answers, a social graph is constructed and the user is recommended to have different levels of access to her profile for users in different groups. This topology-based approach on privacy, by sharing a different profile with different groups, e.g. friends, and friends of friends, has been shown to have its risks as well [Carminati et al. 2011]. In [Carminati et al. 2011], the authors quantify the probability of Unauthorized Access Risk that information is propagated from friends of a user to unauthorized users.

More similar to our approach, is the one of FaceCloak [Luo et al. 2009]. FaceCloak is a Firefox extension allowing users to specify which data in their profile, or Facebook activity, need to be kept private. The sensitive data are substituted with fake ones, while the encrypted version of the private data are published on a third party server. Users email keys to each other, so that their respective FaceCloak extensions can retrieve the real text from the third party server. Then, each time a user retrieves a friend's profile from Facebook, in order to substitute the fake data with the real ones FaceCloak must retrieve also the encrypted version of the sensitive data from the third party server. Hence, FaceCloak relies upon the availability of a "parallel" centralized infrastructure to store the encrypted data rather than leveraging on the existing Facebook platform. An obvious drawback of this solution is that users have to trust the security of the third party server. Also, the third party server represents a single point of failure.

A similar approach has been taken by the authors in [Beato et al. 2011]. The proposed Scramble system is implemented as a Firefox extension that allows users to define access control lists for the data they publish on various popular social network sites. The content, namely, wall posts, private messages and news status, is encrypted with OpenPGP and later on decrypted by the extension under the keys of the group members who are allowed to view that piece of content. The user selects the text that she wants to encrypt and then the extension publishes an encrypted version that can be seen only by the specified friends. A rather similar concept, but implemented specifically for Twitter is presented in [De Cristofaro et al. 2011]. Hummingbird allows users to define access lists for their tweets, which are encrypted before publishing as well as to privately follow hash-tags for the topics of their interest.

NOYB ("None Of Your Business") [Guha et al. 2008] is another system for privacy protection on Facebook. Private profile information is divided into atoms (e.g. name and gender constitute one atom), and each atom is "replaced" with a corresponding atom from another randomly selected user who uses NOYB. Only authorized users can reverse the process of attributing the atoms of information to the real profile. NOYB is implemented as a browser Add-on for the Firefox web browser. The problem with NOYB is that the anonymity depends on the number of its users. Another Firefox Extension is proposed in [Kumari et al. 2011] to enforce user requirements on access level for her data. More specifically, the extension controls events such as print, copy, paste, save, etc. on rendered text and images based on user generated access policies on the server side. The extension is tested through a sample social network, Scuta, which again is not solving the issue for Facebook users.

None of the work in the literature mention the concept of Virtual Private Social Networks (VPSNs), apart from [Figueiredo et al. 2008] that somehow put together the concept of VPN and the one of Social Network. However, we argue that the proposal in [Figueiredo et al. 2008] does not actually bring the concept of VPN into Social Networks (i.e. building a virtual social network over

a real social network). Rather, they propose just another new social network—with privacy features, indeed—but with no relation to a pre-existing Social Network. In particular, they create a new social network over their previously proposed IPOP framework [Ganguly et al. 2006]—that builds an IP network over P2P, hence resulting in a distributed VPN. While the scope of our work is far from building a proper access control model, our aim is similar to the one of access control models for web services [Paci et al. 2011], where access to information is restricted based on user's credentials, and possibly limiting to the user the knowledge about the access policies the user is subject to.

Most of the reviewed literature does not address the problem of publicly available information on Facebook. The few proposed solutions that we mentioned above either (i) rely upon an additional centralized infrastructure or (ii) rely upon the collaboration of Facebook and add unnecessary encryption overhead in users' interaction with Facebook. In contrast, the solution proposed in this paper does not suffer from these limitations.

To the best of our knowledge the concept of Virtual Private Social Networks (VPSNs), as a social network with privacy features build on top of an existing social network, has been introduced for the first time in the preliminary version of this paper [Conti et al. 2011]. In this work, we propose a complete implementation of the VPSN concept in Facebook. In particular, we address an important problem that was left open in [Conti et al. 2011], that is the automatic propagation of a modified profile to all the users within the VPSN. The proposed solution meets the desirable characteristics a VPSN should have [Conti et al. 2011], while being transparent to the user. The implementation of the automatic update is also resilient to servers experiencing failures or behaving maliciously.

3. VPSNS: VIRTUAL PRIVATE SOCIAL NETWORKS

In this section, we recall the concept of Virtual Private Social Networks (VPSNs) we introduced in [Conti et al. 2011], and describe their security features and properties. The concept of VPSNs has some similarities with the concept of Virtual Private Networks (VPNs)—even though they cannot be mapped 1-to-1 to each other: both leverage an underlying infrastructure to build a network which is hidden (to some extent) to the underlying one. Apart from this similarity, we expect a VPSN to enjoy the following features:

— **Virtual.** A VPSN should not have its own infrastructure. Instead, a VPSN has to leverage the architecture and the infrastructure of an already existing social network.
— **Private.** The users that are not part of the VPSN should not be able to access profile information shared within the VPSN. This information must be confidential even with regard to the host social network manager (e.g. Facebook).
— **A social network.** The VPSN build on top of a real social network (e.g. Facebook) should remain a social network itself (as defined in [Boyd and Ellison 2007] and extended in [Cutillo et al. 2009]). That is, it should have the same functionalities of the social network it leverages, even if the shared information is restricted to the VPSN member. In particular, we do not expect the host social network to be used only as a repository, just as we do not expect the VPSN actually being a complete social network infrastructure.
— **Hidden to users outside the VPSN.** The users that are not part of a VPSN should not be able to know about the existence of the VPSN (hence not even the VPSN members can be identified as so).
— **Hidden to the host social network manager.** SNSs managers might not encourage the use of VPSNs, because of the reasons exposed in the Introduction. For example, VPSNs would reduce the benefit of targeted advertisement (e.g. directed to profiles claiming a given current city). Hence, we envisage VPSNs to be as much as possible hidden to the SNSs managers. In particular, VPSNs should not be identifiable by the SNSs, and of course they should not depend on the SNSs collaboration.
— **Transparent.** Once within the VPSN, the users should be able to use the service in a transparent way. That is, a user should be able to browse the real profile of a friend, as it would happen if they were both on the host social network, and sharing there the real profile information. This feature

does not include the features which are a direct consequence of the fact that the user wants indeed to remain private with respect to the SNS, other users and other parties not explicitly authorized (including telco operator, application developers, and mail servers).

— **User might connect to more than one VPSN.** Similarly as it happens for VPNs, in VPSN we expect to be able to partake in several VPSNs. In particular, the user might: i) appear in two (or more) VPSNs with different profiles (e.g. appearing with different profile pictures); ii) appear in different VPSNs with different levels of privacy (allowing friends in the first VPSN to see her current city, while retaining this information from friends in the second VPSN).
— **View.** We also extend the previous concept of a user being connected to different VPSNs, allowing even more fine tuning within the same network. In fact, the concept of appearing with different profiles to different sets of people can be generalized—a user might appear with a different profile (or with a different level of availability of information) to each single friend.
— **View evolution.** The view that one user has of another user's profile might evolve. That is, the amount and type of information shared might change over time.
— **Lightweight.** The infrastructure required to let the VPSN work should be negligible and not be comparable to the one of the host social network. The induced overhead in terms of memory and computation should be as trivial as possible.
— **Heterogeneous.** In the same social network should be possible to have both profiles outside VPSN and profiles using VPSNs. Also, a user should be able to browse, at the same time (e.g. combined in the same web page), both information about friends outside VPSNs and information about friends within the same VPSNs she is connected to.

We observe that VPSNs cannot be easily implemented with the tools currently available in SNSs. One thing that might resemble a VPSN are the Facebook "groups" (or similar concept for other SNSs). However, a group is known to the SNS and it depends on the SNS collaboration. Even if we ignore this fact, there are other two major differences between a "group" and a VPSN. First, groups are centrally managed while VPSN management is completely distributed. Second, groups have a simple access control rule. Once part of the group, all the information within the group are available to the new member. On the other hand, with VPSN it is possible to create different "views" inside the same VPSN.

We underline that the focus of this work is about the privacy of (public) information shared as part of a social network profile. Other aspects connected to the users identification (e.g. traffic analysis), not based on this information, are orthogonal to the scope of this paper.

4. FACEVPSN: OUR PROPOSAL FOR VPSNS IN FACEBOOK

In this section, we give an overview of FaceVPSN, which is our VPSN implementation for Facebook. The design of FaceVPSN is presented in Section 5, whereas the implementation details are discussed in Section 6.

FaceVPSN is also the first solution to address the problem of availability of public information—not only with respect to other users, but also against third party applications in Facebook. The concern for the availability of this information to third party application is confirmed by the fact that Facebook in May, 2010 [Facebook 2012a] gave the user the possibility to completely turn off the applications platform (before these changes, third party applications were able to access by default user's "publicly available information" that consisted of Name, Profile Picture, Gender, Current City, Networks, Friend List, and Pages). Nevertheless, if a user has not turned off the applications platform, Name, Profile Picture, Gender, Networks, and User ID are still publicly available by default.

The availability of this option is undermined by the evidence that *"every month, more than 50% of Facebook users engage with platform applications"* [Facebook 2012a]. Hence, users do not just forget to turn off applications platform, rather they explicitly want and use applications. We consider 50% of Facebook users to be a high enough number for the privacy problem to be considered an unresolved issue. In addition, the availability of profile information to applications

raises an even higher concern. In fact, applications can easily retrieve and process a big amount of profile information. Finally, we note that even if users choose to turn off applications platform, public pieces of information are still available just by browsing profiles, e.g. with `USER_ID` (`www.facebook.com/index.php?profile=USER_ID`), not mentioning the risk [ArchiveExploits 2012] of data stored by Facebook being hacked. Also, we underline that according to Facebook Privacy Policy, Facebook is *"not responsible for third party circumvention of any privacy settings or security measures on Facebook"*.

The simple solution of encrypting the attributes would not work, since Facebook does not allow publication of ciphertext or even scrambled text. Our approach is the following: we cannot stop the sharing of public information due to the design of Facebook, however we can publish pseudo information, which inevitably can be seen by third party applications. According to [Young and Quan-Haase 2009], this is one of the less frequently used privacy protection strategies by Facebook users. This behaviour is actually motivated by the following need: allowing the Facebook friends of a user to see her real Name, Profile Picture, Gender and Current City.

Given that User ID, Network names, Friend's names, and the names of the Pages a User Is-Fan-Of cannot be changed by the user in any way, the real concern is with regards to Name, Profile Picture, Gender and Current City. Hence, we propose that those fields are modified, through the Profile Editor of Facebook, in order not to contain the real values that correspond to Facebook account owner: we suggest to use pseudo values instead. This modification should be done by the user herself for three reasons:

(1) The user can select the pseudo information to display, and thus still be in control of her published profile page (e.g. how her pseudo profile picture will look);
(2) The user can choose not to share some of the real details with certain friends;
(3) Users are already familiar with modifying profile information, hence doing this modification is not a barrier for using our proposed system.

Three issues are left open at this point: (i) an authorized user being able to see real information while browsing pages on Facebook; (ii) how the real information is sent to friends; (iii) how friends are later on notified about updates of real information. For issue (i), once pseudo information is stored on Facebook, the real information is sent only to friends allowed to see it. This information will be stored locally on the friends' machines, with a mechanism that is similar to how cookies work in browsing. However, differently from cookies, once this information is stored locally, it will not be sent back on a later occasion. Instead, it will be processed locally, when required. When a user browses a profile of another user in the VPSN, a FaceVPSN component is in charge to transparently show to the user the real information, instead of the one actually published on the host social network. Point (ii) requires two users of FaceVPSN (that want to become friends) to exchange some information outside Facebook and FaceVPSN—we argue that a direct involvement of both users is mandatory since we want to avoid: centralized solutions; publicly available information; as well as we do not want to reveal that a Facebook user is actually using FaceVPSN.

As for (iii), a component of FaceVPSN will be in charge to automatically propagate an updated profile to all the interested friends. While completely automated and transparent to the user, the communication mechanism used by FaceVPSN is basically the same used to deliver users chat messages (e.g. for Gtalk). The FaceVPSN component of the profile owner will publish the update and the FaceVPSN of all her friends that were subscribed to her profile will receive automatically the update by means of the pub/sub system. In particular, this propagation of profile updates will is done via the publisher/subscriber service (XMPP PubSub) [Foundation 2012] implemented by the Extensible Messaging and Presence Protocol (XMPP) [Saint-Andre 2011a][Saint-Andre 2011b]. XMPP is an application profile of XML that enables the near-real-time exchange of structured data (called "stanzas") between any two or more network entities. XMPP provides a technology for the asynchronous, end-to-end exchange of stanzas by means of direct, persistent XML streams among a distributed network of globally addressable, presence-aware clients and servers. The XMPP network uses a client-server architecture (clients do not talk directly to one another but by means

of the server networks). XMPP is extensible by design and the most relevant extension for our work is the XMPP PubSub system [P. Millard 2010]. XMPP PubSub can implement a content-based pub/sub. In such pub/subs, the set of interested subscribers is determined at runtime based on the content of publications, which is typically represented by a header containing a collection of attributes and associated values. Subscriptions identify publications of interest via predicates that specify constraints over these attributes. We use this infrastructure to deliver our profile updates.

The implementation of XMPP PubSub provides natively, support for recovery in case of link failure and XMPP server failure (e.g. by clustering thus enabling a single domain to be supported by multiple servers [isode.com 2012]). In particular, to deal with stream management and reliability issues, XMPP implements two features: Stanza Acknowledgments – the ability to know if a stanza or series of stanzas has been received by one's peer and Stream Resumption – the ability to quickly resume a stream that has been terminated. It can happen that an XML stream is terminated unexpectedly (e.g., because of network outages). In this case, it is desirable to quickly resume the former stream rather than complete the tedious process of a new stream establishment. In addition, this protocol exchanges the sequence numbers of the last received stanzas on the previous connection, allowing entities to establish definitively which stanzas require retransmission and which do not, eliminating duplication through replay.

Furthermore, to be resilient to malicious XMPP PubSub server, users can register to multiple XMPP PubSub servers at the same time—note that there are many XMPP servers available (example of lists can be found here [xmpp.org 2011; jabberes.org 2011; thecoccinella.org 2011]). Finally, using multiple XMPP PubSub servers, the overall solution remains not dependent on the collaboration from Facebook, as well as not from any specific party. The features we require from the XMPP PubSub (e.g. handling events for users that are currently off line) are specified in Section 6.1.

5. FACEVPSN DESIGN

FaceVPSN is implemented as a Firefox extension, made up of two main logical components: (i) the *Replacer* that handles the actual browsing of private profiles over the public Facebook network; (ii) the *ProfileUpdater* that handles the propagation of a user profile update within the VPSN.

To describe the architecture and the behaviour of FaceVPSN, we use the notation in Figure 1.

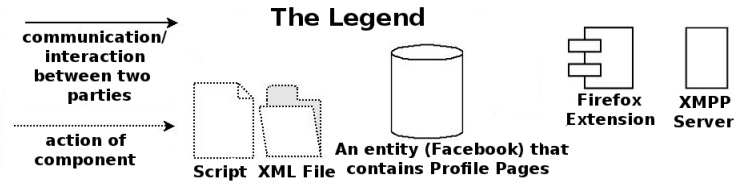


Fig. 1. Graphical Notation.

A general overview of the interactions in FaceVPSN is given in Figure 2. First, Facebook User 1 would modify her Name and Profile Picture to some pseudo information through the profile editing interface of Facebook (Step 1). In order to partake in a VPSN, a user must have FaceVPSN installed. Hence, in a practical scenario, if User 2 is new to the VPSN, User 1 might ask her to install FaceVPSN (Step 2). Also, the two users need to exchange some initial information (Step 3). User 2 has to add the XML file corresponding to User 1 to the list of preferences. This is done through the Graphical User Interface (GUI) of FaceVPSN.

For ease of exposition, in the remaining part of the paper we will consider the simple scenario where the user who wants to use FaceVPSN does not have yet a Facebook account. However, in practice there are several scenarios possible, and for each of them there are different possible approaches to use FaceVPSN:

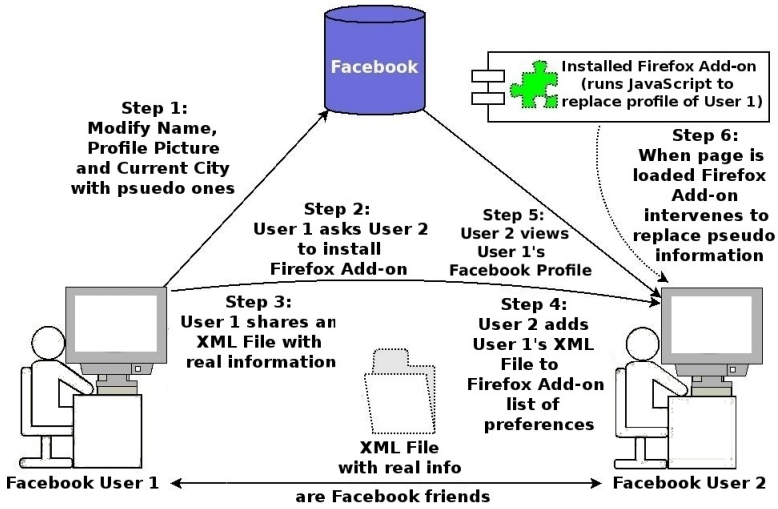


Fig. 2. Overview.

— No existing profile. The user will just start using Facebook and FaceVPSN at the same time. The user will choose the pseudo information to be registered on Facebook.
— Existing profile, already with pseudo information published. In this scenario (which seems to apply to some of current Facebook users) the user (via FaceVPSN) can use the same pseudo information to create XML files to be distributed to her friends. We observe that in this scenario, the privacy of the user was already protected as it will be with FaceVPSN, however the user and her friends will significantly benefit from FaceVPSN in terms of usability—they will be able to browse the information as if the user did put her real information on Facebook.
— Existing profiles with real information. In this case, we believe that the best approach is that the user creates a completely new account and asks again their friends to join her profile. As another option, the user might opt for modifying her published real information with pseudo ones. While this solutions might be more usable—no need to ask friends to connect to the new profile—Facebook would still be able to link the new pseudo information to the previous one.

5.1. Set Up

In order to partake in the VPSN, the user needs to install FaceVPSN Firefox extension, which currently works only with Mozilla Firefox web browser. FaceVPSN is installed just like any other Firefox extension. It is as simple as dragging the downloaded XPI installation file to Firefox browser and a menu will pop up to confirm installation. As part of the installation process, the ProfileUpdater component of FaceVPSN has to perform some specific actions. These are described in Figure 3. In particular, during the installation of the Add-on (Step 1), FaceVPSN creates a node (topic) identified by her account on each of the predefined N XMPP PubSub servers (this is shown in Step 2; where N is a system parameter, e.g. N=5). As a result, each user that installed FaceVPSN will have a topic (account) on each of the N XMPP PubSub servers. For each user, the topic information for each pub/sub server is then stored in a file called ProfileUpdaterInfo.xml (Step 3). The information related to these topics (accounts) will be used when creating what we call the *FaceVPSN business card*—which usage is described later—, and to propagate to the friends the updated XML profile.

Figure 4 details the process of exchanging information between two users that want to be in the same VPSN. First, User 1 shares her FaceVPSN business card with User 2 (Step 1); the business card contains the following information:

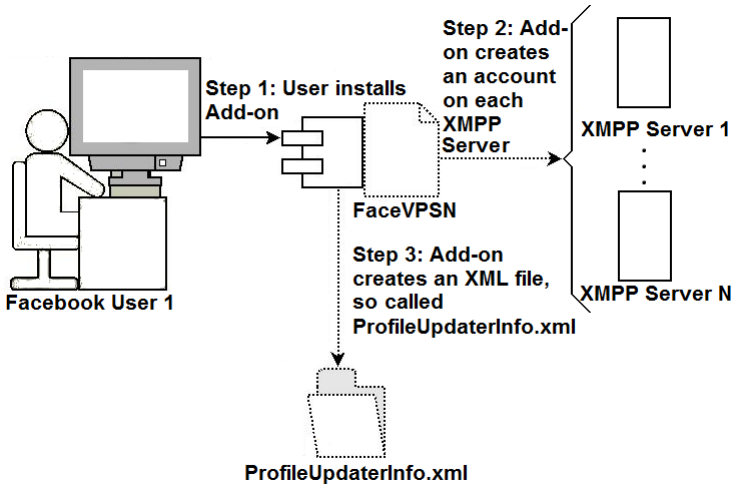


Fig. 3. Setup of ProfileUpdater component.

— pseudo name with which User 1 is registered on Facebook;
— an XML profile containing the real information of User 1;
— the symmetric key used to encrypt further updated XML profile sent from User 1 to User 2 (the 256-bits key is randomly generated by User 1);
— a set of n XMPP PubSub server IDs (randomly selected out of the N predefined servers), and the corresponding topic names (accounts) of User 1 on these n servers.

The FaceVPSN module on User 2 then subscribes to the topics specified on the n servers selected in the business card (Step 2).

We remind the reader that these accounts were created during the installation of FaceVPSN. Hence, at this step, it is just matter of selecting information from a list. Finally, FaceVPSN stores the exchanged information in ProfileUpdaterInfo.xml (Step 3). We observe that since this exchange involves the transmission of confidential information (XML with real information, and symmetric key used later on), this communication must take place over a trusted channel. For example, this might happen via email (possibly encrypted) or via seeing-is-believing approach [McCune et al. 2005]. In the latter case a practical exchange can happen as follows: when two users meet in person, User 1 uses a FaceVPSN application on her mobile phone to generate a QR-code containing her business card. In turn, User 2 can scan this QR-code with her mobile phone, generate the answer, and transmit it back via the same mechanism. Because of the typical usage of mobile phones (as personal devices), these would also be the ideal candidates to act as a portable storage where the XML profiles are stored. No matter the computer machine where the user will log in to browse Facebook, the XML profiles for her VPSNs will be always available with her, and accessible (e.g. via Bluetooth, or Near Field Communication) by the FaceVPSN component installed on the computer machine.

A variant solution to the storage issue has been also envisaged in [Conti et al. 2011], i.e. steganographing data within images uploaded on Facebook.

5.2. Browsing

Once FaceVPSN is installed on the users machine and the proper information are exchanged, all that remains during browsing is to replace the pseudo information of Facebook User 1 with real information of User 1, whenever User 2 views User 1's profile. A similar replacement also happens when User 2 browses any other page that contains User 1's information (Step 5 and 6 in Figure 2).

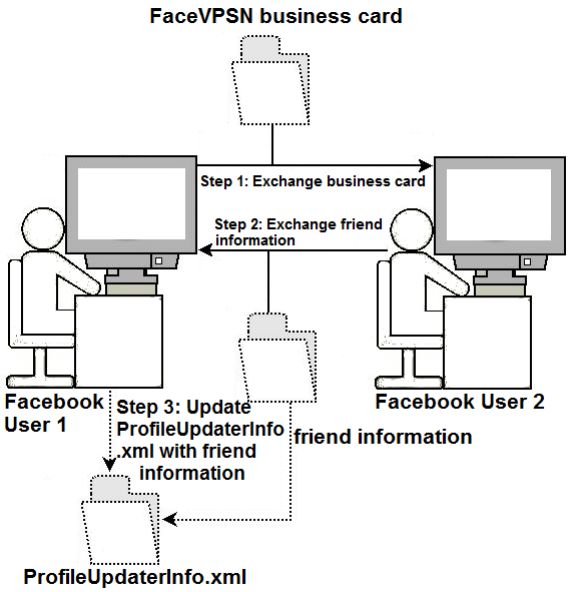


Fig. 4. Initial information sharing.

More in detail, after a Facebook page is requested (Step 1 in Figure 5), Facebook replies with an HTML response that contains the content of the page to be displayed by the Firefox browser (Step 2). The replacing functionality of FaceVPSN is implemented in JavaScript code, which is executed once certain events happen. In particular, when the document is loaded from Facebook, a "page load" event is fired (Step 3). Then, the JavaScript code of FaceVPSN searches to find pseudo names and profile pictures of the friends of the user. Afterwards, the JavaScript code replaces them with the real information stored in XML files. In the end, the Firefox browser displays the profile page with replaced pseudo information (Step 4). FaceVPSN replaces Name and both sizes of the Profile Picture, the big one shown in the profile page and the small one shown in all the other pages. The application is designed in such a way that it is also easily extendable to handle the replacement of other profile information.

5.3. XML Profile Update

The behaviour of FaceVPSN when User 1 updates her profile is as described in the following. The solution for this problem exists in two parts. The first part is to identify that an XML has been updated, which can be as simple as reacting to a click on a button. Then, we need to format the XML in such a way that it can be sent through a XMPP PubSub servers to all the friends registered for that XML—that are using that XML as their "view" on the profile of User 1. Finally, it needs to actually publish the formatted XML. For the second part, on the other side (friends of User 1), FaceVPSN receives the update by means of subscription to User 1, extract the formatted XML, and update the local XML accordingly.

This process of propagating the updated XML is shown in Figure 6. First, User 1 updates her local XML file containing her data (Step 1). FaceVPSN sees this change, and processes the XML file as a message (Step 2), than it publishes the message through an XMPP PubSub server (Step 3). This concludes all the work that FaceVPSN has to do on User 1 side. We stress that the update is completely automated and transparent for both users: for User 1 that is modifying her profile information, and for her friend User 2 that will browse the updated profile. No matter how many

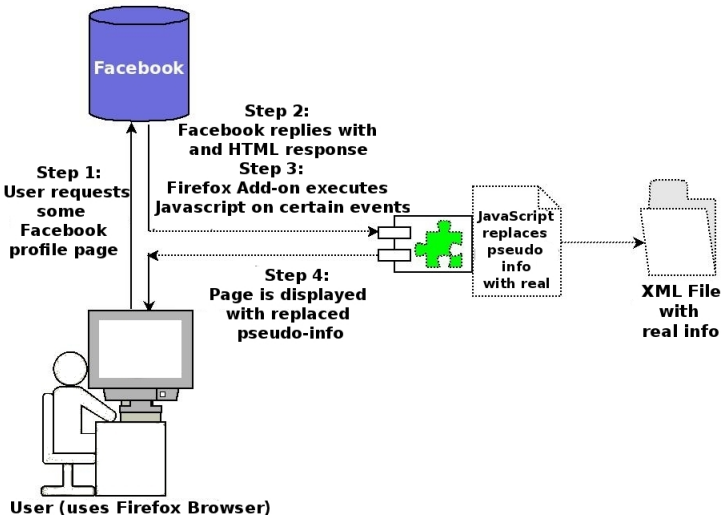


Fig. 5. Browsing.

friends User 1 has, her profile updates will be (automatically and transparently) propagated by FaceVPSN to all of her friends. When User 2 browses the page of User 1, FaceVPSN starts to work (Step 4). In particular, the role of the ProfileUpdater component is here also to get potentially updated profiles info available on XMPP PubSub servers (Step 5). If a new XML profile is received for User 1 (Step 6), FaceVPSN checks its encryption and its counter number. If the message is properly received and decrypted, and its counter is higher than the counter of the XML stored locally for User 1, FaceVPSN replaces the current local XML for User 1 (Step 7). If such an update succeeds, the ProfileUpdater component asks the Replacer component to replace the information on the profile page currently shown on Firefox.

Since, the update is replicated on different XMPP PubSub servers for redundancy and most importantly for security, so that no single compromised server can inhibit our protocol, the FaceVPSN of User 2 takes care also of dealing with multiple copies of the same update by simply considering the first one and dropping the others. We recall that all copies of the same update have the same version number.

5.3.1. Scalability. Profile update is an operation that users do from time to time rather than daily. Thus, the traffic generated by such updates is expected to grow linearly (factor a multiplicative constant) with the number of users adopting FaceVPSN. Nevertheless, if most of Facebook's users will use FaceVPSN, scalability is an issue to consider. By adopting a publisher/subscriber system as event delivery mechanism, we took already this issue in consideration, since such systems were introduced with the aim of being scalable. Furthermore, even if the primary goal of the actual XMPP PubSub design is not to build an Internet-scale system, publisher/subscriber specifically designed for scalability and performance have been already proposed (e.g. SCRIBE [Rowstron et al. 2001], SIENA [Carzaniga et al. 2001]) and can be used by FaceVPSN.

6. FACEVPSN IMPLEMENTATION

FaceVPSN has been implemented as a Firefox extension and its functionalities are written in JavaScript, but the functionalities of the ProfileUpdater component have been implemented via PHP scripts using XMPPHP library. The Firefox extension and the PHP scripts interact via the following mechanism. The Firefox extension can observe what are the pages browsed by the user, and also implements the Replacer component. When required, the Firefox extension invokes the locally

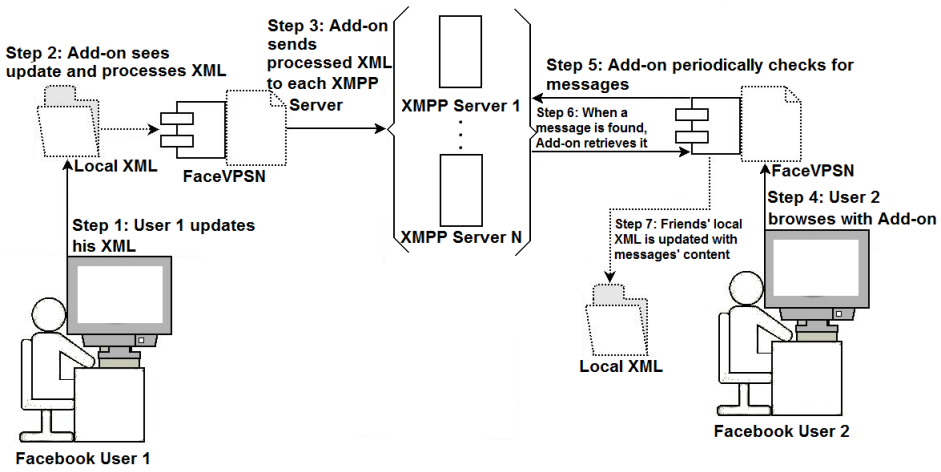


Fig. 6. Propagation of the updated XML profile.

stored PHP scripts (via the Mozilla XPCOM interface `nsIProcess` [Mozilla 2012e]). We remind that these scripts implement the functionalities of the ProfileUpdater: i) create a topic and publish a modified user-profile to the pub/sub server; ii) subscribing to topic representing friends' accounts iii) getting available updated profiles. Observe that the user does not need to explicitly interact to run those scripts. The installation of PHP CLI (that does not require having a local or remote web-server to execute PHP) will be done together with the installation of FaceVPSN. The GUI of FaceVPSN has been written with XUL, the Firefox XML user interface markup language.

6.1. Set Up

When FaceVPSN is installed, the ProfileUpdater component takes care or creating N different topics (named with the same random account identifying the user.) on a pool of N XMPP PubSub servers—an updated XML profile will be communicated to a friend using a subset of n out of the N servers. The information about the created accounts (i.e., for each server: server name, topic, and the password) will be stored locally in a XML file (servers.xml).

We recall that there are many XMPP servers available [xmpp.org 2011; jabberes.org 2011; thecoccinella.org 2011]). Also, note that for our purposes the servers should preferably be chosen to have the the following two characteristics: i) the creation of a new account should be possible via a script, i.e. without requiring any active user participation, like solving CAPTCHAs; ii) they should be able to handle messages for off-line parties. An example of servers with such characteristics are the ones selected for our experimental evaluation (listed on x-axis of Figure 14(a)). The first characteristic is there to allow the script to create the accounts without any user interaction. The latter requisite is there to allow to transmit updated XML profile even to users that are off-line when the update is made (this might be quite a common scenario). In this way, the FaceVPSN component that publishes the updated XML does not need to take care of the status (on-line/off-line) of the receiving friend. Finally, note that even if the two requisites are desirable, point i) might be relaxed requiring a bit of interaction from the user (e.g. solving CAPTCHAs, or filling form)—only at installation time, and ii) might be relaxed at the cost of a higher complexity in the management of the transmission and probably a higher delay in the propagation of the updated profile—i.e. the transmission can happen only when both users are on-line at the same time, or leveraging a multihop transmission via common friends.

When two friends want to be connected on Facebook, they need to share the information about the $n < N$ XMPP servers they will be using to update their XML profiles. The user that initiates

```
<ProfileUpdaterInfo>
 <file>
  <name>"XML Profile 1"</name>
  <personalInformation>
  ...
  </personalInformation>
  <friendList>
   <friend>
    <encryptionKey>SymKey</encryptionKey>
    <server>
     <address>"xmppserver_1"</address>
     <topic>"topic_1"</topic>
    </server>
    ...
    <server>
     <address>"xmppserver_n"</address>
     <topic>"topic_n"</topic>
    </server>
   </friend>
  </friendList>
 </file>
</ProfileUpdaterInfo>
```

Fig. 7. ProfileUpdaterInfo.xml.

```
<FaceVPSN version="0.2">
 <friend>
  <urls>
    <username>facebook.username</username>
    <id>user-ID-00000</id>
  </urls>
  <profile-counter>4</profile-counter>
  <replacementinfo>
    <info>
      <pseudoname>"Pseudo Name"
      </pseudoname>
      <realname>"Real Name"</realname>
      <pseudopicurl>
        "http://pseudoSmall.jpg"
      </pseudopicurl>
      <realpicurl>
        "http://realSmall.jpg"
      </realpicurl>
      <pseudobigpicurl>
        "http://pseudoBig.jpg"
      </pseudobigpicurl>
      <realbigpicurl>
        "http://realBig.jpg"
      </realbigpicurl>
    </info>
  </replacementinfo>
 </friend>
</FaceVPSN>
```

Fig. 8. Structure of XML profile.

the procedure sends to the other user a FaceVPSN business card, containing the information as described in Section 5.1, and implemented via an XML file structure as shown in Figure 9. In case the pairing take place with smartphones, as envisaged in Section 5.1, Figure 10 shows the QR-code corresponding to the FaceVPSN business card in Figure 9.

```
<FaceVPSNBusinessCard>
 <facebookInformation>
  <username>facebook.username</username>
  <id>user-ID-00000</id>
  <initialXMLReplacer>
  ...
  </initialXMLReplacer>
  <encryptionKey>SymKey</encriptionKey>
 </facebookInformation>
 <servers>
  <server>
   <address>"xmppserver_1"</address>
   <topic>"topic_1"</topic>
  </server>
  ...
  <server>
   <address>"xmppserver_n"</address>
   <topic>"topic_n"</topic>
  </server>
 </servers>
</FaceVPSNBusinessCard>
```

Fig. 9. FaceVPSN business card.

Fig. 10. QR-code for FaceVPSN business card in Figure 9.

Once the other friend sends back the required information to join the VPSN, i.e. her account name on the proposed n servers (together with the info in her FaceVPSN business card), FaceVPSN on User 1's machine will locally store these information in the file ProfileUpdaterInfo.xml (Figure 7). This file contains, for each XML profile a `<file>` entry. This entry contains the list of friends that

must be updated when the XML profile file is updated. In turn, for each friend, the file contains the information about the symmetric key to be used to encrypt data for that friend, and the servers and account to be used to transmit the data.

6.2. Browsing

In order to share real information in a structured way, which will be useful and easily accessible by the Firefox extension, we propose to use an XML file with the structure of the example shown in Figure 8. The user is not required to fill in every field. Nevertheless, she must specify at least her username and user ID. Then, for instance, she can choose to have only pseudo Name and not the Small or Big Profile Picture.

Essentially, everything is contained under the root tag `<FaceVPSN>`. First, we have a version number in order to track changes of structure of the file (compared to the version presented in [Conti et al. 2011], version *"v0.2"* includes a profile counter, to handle updates of the profile). In Figure 8 we show the typical case, that is, when a user shares only her information. Nonetheless, it is also possible that two or more users can share information in the same XML file; hence using more `<friend>` tags—the information of one person is contained inside those tags. We believe that the choice of using this implementation is convenient, for instance, when only one of close friends or siblings has to take care of sharing the file for two or more people. Moving down the XML hierarchy, the reader can observe that we classified information into two categories:

— information contained inside `<urls>` tags, which will be used to check URLs to verify to whom the link or profile page corresponds to. This includes user's Facebook username (contained inside `<username>` tags) and user ID (contained inside `<id>` tags);
— information that will be used to replace pseudo information, and contained inside `<replacementinfo>` tags. This category of information contains the pseudo Name and the corresponding real value. Additionally, there are two different Profile Picture URLs in Facebook. The most frequently used is the small boxed one, that appears for example in Wall posts. Yet, there is also a bigger profile picture that appears in a user's profile page with a different URL.

Users add through FaceVPSN's GUI the XML files that refer to their friends—in order for FaceVPSN to know whose details to replace. FaceVPSN uses the preferences system of Mozilla [Mozilla 2012c] in order to save the user's preferences when the browser is started and closed. It also uses the input stream component of Mozilla [Mozilla 2012c] to actually open and read a file that the user has chosen from the dialog window. FaceVPSN handles replacement of pseudo names in text as well as links. Furthermore, it replaces the real name with a pseudo name when searching in Facebook (e.g. when tagging or in the search bar).

The document's URL is checked to find either the friend's ID or username. If either is found, it means the user is browsing a profile page of a friend. In that case, both, the link and text names are replaced. On the other hand, if either the friend's ID or username is not found, the user is browsing in the Facebook home page or some other (e.g. events, messages, photos) page. In this case, FaceVPSN replaces only names that appear in links or next to images of friends' profiles.

Profile pictures are replaced by changing image source to a different source link (remote or local). An optimization was introduced from the previous version of FaceVPSN, presented in [Conti et al. 2011]. In fact, first of all the type of pictures Facebook is using decreased from three types (big version, small version, a medium size picture used in in the functionality to search for friends) to two types (big and small—the small is now used also in the search). Furthermore, we were able to significantly optimize the search time as follows.

In the previous version of FaceVPSN, we were getting a list of images from the current page and then, for each friend, we were checking whether a replacement was needed. In the current version, for each friend (who is using FaceVPSN) we have created an object within a `<key,value>` structure: where the `key` is the URL of the (pseudo) pictures that need to be replaced, and the `value` is the URL (remote or local) of the (real) picture that should replace the `key` in the loaded page. Since we can pre-compute this object for all the friends, the substitution of the pictures in the

loaded page is much faster: we loop only once through the list of retrieved images and search them in the created object. In short, in the previous FaceVPSN implementation, the list of pictures to be replaced was searched in the list of retrieved images from the Facebook pages, whereas now, the list of retrieved images is searched in the precomputed object that is much smaller in size.

In particular, please note that this pre-computation of the object is possible since the URLs (the `key`) pointing to the pictures of a profile are known[1] and do not change over time.

To start the execution of replacement functions we add a Gecko (which is the Firefox rendering engine) specific event (i.e. DOMContentLoaded). This event is fired when the DOM Content of a page is loaded, albeit without any documents or images on it. Nonetheless, when the user navigates in Facebook, FaceVPSN has to re-perform replacements due to new content being displayed. For instance, the URL changes when a new picture of an album is being viewed or a different profile tab is opened. To get notified whenever the URL in the address bar changes, we added a progress listener to our code to re-apply replacements whenever location changes. Progress Listeners [Mozilla 2012c] allow Firefox extensions to be notified of events associated with documents loading on the browser and with tab switching events.

In the current version, Facebook relies heavily on the AJAX technology to refresh the content of the page (in parts or for the whole HTML document). FaceVPSN monitors these events through HTTP Request Observers [Mozilla 2012b], in order to perform replacements of the newly presented information. In particular, FaceVPSN monitors AJAX requests every 500 ms.

More details on the implementation, are available on the project website [Hasani 2012].

In order to have the published (non real) information replaced on the browsed page, FaceVPSN has to match different regular expressions to find them. These are used primarily to find information that needs to be replaced, but also, for instance, if we consider the case of two friends with the same name then the replacements are to be performed based on some regular expression matching. This requirement can generally be fulfilled by distinguishing information based on usernames or user IDs, which are unique.

A further optimization was made possible by the new Facebook layout. More specifically, in the previous version a friend name or ID were appearing in the URLs in different combinations other with other words such as `photos`, `mutual`, `photo_search`, and with other friend ID and names (depending on the interaction the user was having with Facebook). Hence, to make a proper replacement it was necessary to consider 30 different regular expressions and check them against each URL within the facebook loaded page [Conti et al. 2011]. Currently, the structure of the links have been simplified by Facebook to be just either `https://www.facebook.com/friend.id` or `https://www.facebook.com/friend.username` in all the links referring to a specific friend (no matter the specific interaction and browsing path the user is having with Facebook). Hence, searching just two regular expressions (listed in table I), it is possible to always distinguish and replace the information correctly. This constitutes a significant improvement in terms of the time overhead, with respect to the version of Facebook and FaceVPSN considered in [Conti et al. 2011].

Table I. List of Regular Expressions to match the name in several places.

Number	Regular Expression	Matches links with
1	`afriend.username`	username (e.g. name.surname)
2	`afriend.id`	id (e.g. 11111111)

Of course, the defined regular expressions are SNS dependent. That is, if the Facebook implementation change they must be adapted. For this, we envisage FaceVPSN to be continuously supported

[1] E.g. for our test profile:
`https://fbcdn-profile-a.akamaihd.net/hprofile-ak-snc4/41496_100000523696574_9088_q.jpg` and `https://fbcdn-profile-a.akamaihd.net/hprofile-ak-snc4/41496_100000523696574_9088_n.jpg` point to the small and the big picture, respectively.

(e.g. by the open source community) and to automatically check for updates. Similarly, the regular expressions must be adapted if we aim at designing a VPSN implementation for other SNSs (e.g. Orkut), while the same general concept can be applied.

6.3. XML Profile Update

When a user, say User 1, changes her profile information, the changes are reflected in the corresponding XML file stored locally. At this stage, the ProfileUpdater component of FaceVPSN has to take care of the automatic publishing of the updated XML. In particular, as soon as the profile is modified, a proper script is called on User 1 machine to publish the modified XML to the friends listed in ProfileUpdaterInfo.xml (shown in Figure 7). On the side, friends of User 1, e.g. User 2, as soon as she browses the profile of User 1 the ProfileUpdater component receives the most updated profile (if any) to be shown. Note that, on the retrieving side, even if the XML are sent to $n < N$ servers, as soon as the first message is properly received (i.e. a file is available, its decryption succeeds, and the counter of the received XML profile is higher than the counter of the stored XML profile), the local XML profile is updated and the proper real information are shown to the users.

7. EVALUATION AND DISCUSSION

The implementation of FaceVPSN shows that it is indeed feasible to actualize the concept of VPSN. Furthermore, for the experiments shown in this paper, we upgraded the FaceVPSN implementation from the previous version used in [Conti et al. 2011] (that was for Firefox 3.6.8, and the Facebook layout of November 2010) to the a later version of Firefox (ver. 14.0.1) and considered the currently available features of Facebook (December 2012). The fact that this upgrade did not take much effort, further demonstrates the feasibility of our proposal requiring a small maintenance team. However, an automatic application update feature on the application side would be desirable. Apart from these feasibility considerations, in this section we also give a thorough evaluation and discussion of FaceVPSN. First, we briefly discuss the security of the scheme (Section 7.1), then we compare FaceVPSN with other solutions in the literature (Section 7.2) for mitigation of privacy threats in Facebook. Hence, we do not consider here solutions proposing new Social Networks independent from Facebook—our aim is indeed to preserve the privacy of some 1 billion users already using Facebook, rather than convincing them to use a new social network. Further on, in Section 7.3 we discuss the results of experimental evaluation we performed with FaceVPSN.

7.1. Security

As for the security of the scheme, we underline that all the replacement is based on:

— The pseudo information stored in Facebook and the one sent by Facebook to the browser, in reply to a specific request of the user;
— XML files that contain the information for the replacement of pseudo information with real ones.

We assume the first XML is shared between two users on a secure channel. The first XML also contains the information to authenticate the publisher of following updates for that XML files (e.g. a public key certificate of the sender). It is the responsibility of the creator of the XML files to put there correct information about herself: putting in the XML files fake information is simply what a user can already do without FaceVPSN. Similarly, Facebook might sent wrong (pseudo) information to FaceVPSN to have it make a wrong replacement: e.g. force FaceVPSN to display information about Alice, while the user is browsing Bob's profile. However, also this second attack is already possible with the current Facebook (without FaceVPSN). Hence, since both attacks are already possibile without FaceVPSN, FaceVPSN does not decrease the security of Facebook, while it significantly increases users' privacy.

One might think that our scheme introduces new weaknesses that could be exploited by an adversary to run a new attack, i.e.: a malicious user might want to modify the information that another user (say User 2) sees about a victim (User 1). We argue that this attack is not possible, thanks to the security enforced in two moments: (i) during the set-up or the friendship between User 1 and

User 2 (the set-up is discussed in Section 5.1), and (ii) during the updates of the XML file. As for (i), we assume the first exchange of XML is done in a secure way, in particular, in Section 5.1 we cited two possible scenarios (encrypted emails or via seeing-is-believing approach). Furthermore, the *FaceVPSN business card* that User 1 shares with User 2 during the set-up contains a symmetric key used to encrypt the following updated XML profiles sent from User 1 to User 2 (the 256-bits key is randomly generated by User 1). As for (ii), since following updates will be encrypted, an adversary cannot forge fake information for User 1. Hence, to break the scheme the adversary would need to guess a 256-bits key, which is currently considered not feasible for practical scenarios.

7.2. Comparison

Unlike other solutions, FaceVPSN is a decentralized solution that does not depend on the collaboration of Facebook. Moreover, it handles the replacement of the "publicly available information" that the user can modify (i.e. Name and Profile Picture). In particular, FaceVPSN enjoys all at once a set of features that no other current solution has. Table II summarizes the main points we highlighted with regards to the relevant solutions (about mitigating privacy threats in Facebook) in addition to FaceVPSN. The column headers shows five different features, while row headings indicate the considered solutions. In particular, we compare FaceVPSN with: FaceCloak [Luo et al. 2009], flyByNight [Lucas and Borisov 2008], NOYB [Guha et al. 2008], Persona [Baden et al. 2009], and SIGs [Sorniotti and Molva 2010]. These are all solutions directly related to Facebook privacy issues. We remind the reader that our aim is to mitigate the privacy threat of the millions of users using SNSs like Facebook, and building VPSN without having an infrastructure comparable to those of SNSs. Hence, we do not consider solutions like Safebook [Cutillo et al. 2009] and PeerSoN [Vu et al. 2009]. In fact, these solutions do not mitigate privacy in Facebook but rather propose new competitor SNSs.

Table II. Comparing FaceVPSN with state of the art solutions for privacy threat mitigation

	Facebook Collab.	*Centralized/ Distributed*	*Overhead*	*Pre-requisites*	*Hide public info*
FaceCloak [Luo et al. 2009]	No	Centralized	1 s (avg) to view real text	Manage keys	Partially
flyByNight [Lucas and Borisov 2008]	Yes	Centralized	depends on message size	Facebook app	Partially
NOYB [Guha et al. 2008]	No	Centralized	no data	Facebook app	Partially
Persona [Baden et al. 2009]	Yes	Distributed	2.3 s (median; max 13.7 s)	Facebook app	No
SIGs [Sorniotti and Molva 2010]	Yes	"thresholdized"	no data	Their proxy	No
FaceVPSN	**No**	**Distributed**	**<1 s (avg) to view initial replacements**	**none**	**Yes**

The first column of Table II (i.e. Facebook Collaboration) indicates if the solution requires the Facebook collaboration in order to work or exist. More specifically, we are interested to know whether Facebook can remove the application from the interaction cycle of users with Facebook. For instance, the solutions flyByNight, Persona, and SIGs require Facebook collaboration. That is, they can be removed by Facebook from the applications directory and consequently cannot be used by Facebook users. As such, the successful solution of the privacy problems by these applications is under direct threat from Facebook itself. However, considering that potentially every implementation aimed at solving privacy issues in Facebook depends on the Facebook implementation, we do not bring this second viewpoint into this column. For instance, if Facebook adds new profile information fields, NOYB has to be extended to support them or, when Facebook API changes, Persona and flyByNight have to be harmonized accordingly. Likewise, when Facebook changes the design of profile pages and how profile information is handled and displayed (e.g. different HTML tags) then FaceVPSN also needs to be adjusted to handle the changes. Nevertheless, we do not consider this aspect a barrier for the existence of a solution.

We also compared the architecture of the solutions (second column of Table II). In particular, we observed whether the solution was implemented in a centralized or distributed architecture. In

a centralized architecture there is one central component which is fundamental to the functionality of the whole system. This type of architecture suffers from the "single point of failure" problem. Conversely, a distributed architecture is one where there is no need for a central component or coordinator for the operation of the system. A third type, that is "thresholdized", applies to SIGs [Sorniotti and Molva 2010]. In particular, in SIGs a group is managed by a set of so-called managers that take decisions (e.g. admitting a new user in the group) using a threshold consensus. That is, before taking a decision, at least a given number (threshold) of managers have to agree on that decision. FaceCloak, that is implemented as a centralized architecture, requires an infrastructure that becomes comparable to that of Facebook (or even worse, when users belong to many VPSNs). Furthermore, in FaceCloak, if the third party server for storing encrypted private data is down, users' FaceCloak extension cannot decrypt information from Facebook. Similarly, if the flyByNight application server is down, users cannot see decrypted messages from their Facebook friends. In its current implementation, NOYB is also centralized in nature, given that the NOYB Firefox extension depends on external public dictionaries that it queries to retrieve and replace "atoms". Persona as a social network proposal is a decentralized architecture: there is no central coordinator and users themselves define policies for sharing information through their "Storage". Nonetheless, the Persona Facebook application, just like flyByNight, depends on a third party server (i.e. the server where the application resides). In general, the centralized architectures with a single point of failure are potentially a barrier to widespread usage in social networks. On the contrary, FaceVPSN is implemented as a Firefox extension that revolves around its users. As it was specified in Section 5, the user herself is in charge of when and what information to share with others. Moreover, if the FaceVPSN Firefox extension of a user fails, this will not affect her friends.

The column labelled as "Overhead" shows the time overhead of the different solutions. To put it differently, if the data is available, it shows the additional time required for a user to see the benefits of the solution. As for previous solutions, the time overhead indicated in Table II is the one reported in the corresponding publication paper [Luo et al. 2009; Lucas and Borisov 2008; Guha et al. 2008; Baden et al. 2009; Sorniotti and Molva 2010]. The overhead in flyByNight, which handles the encryption of private messages in Facebook rather than "publicly available information", depends on the size of the message. Both [Sorniotti and Molva 2010] and [Guha et al. 2008] do not provide information about their overhead, while we argue this is a relevant characteristic of these solutions. NOYB has an overhead that depends on the number of atoms to be replaced, as well as the number of different keys that need to be generated. In turn, this depends on the updates of profile fields and is upper bound by the number of fields. As a distinguishing factor of FaceVPSN from centralized solutions, we formally define the storage overhead of FaceVPSN (distributed) compared with the one of FaceCloak (centralized)—a similar comparison holds for other centralized solutions. Let us define: N as the total number of users; u_i the generic user ($1 \leq i \leq N$); the set of VPSNs the user u_i belongs as $V(u_i) = v_1, \ldots, v_j, \ldots v_V$; $Size(v_j)$ as the number of elements in VPSN v_j. We remind the reader that each user can give a different view (i.e. show a different profile) for every other user in the VPSNs she belongs to. FaceCloak [Luo et al. 2009] requires a storage that is $O(\sum_{i=0}^{N} \sum_{j=0}^{V(u_i)} Size(v_j))$. On the other hand, FaceVPSN (with decentralized architecture) requires a limited local storage (and computation) overhead that is $O(\sum_{j=1}^{V(u_i)} Size(v_j))$, for user u_i.

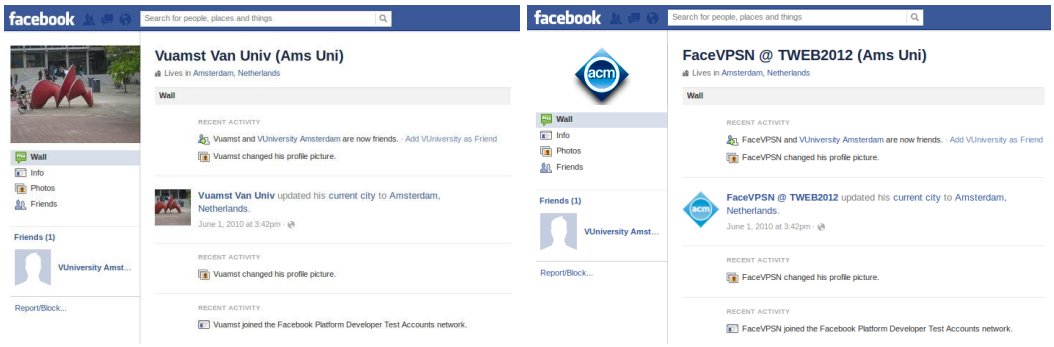
In the "Pre-requisites" column, we are mainly interested in potential barriers to the usage of the system. For instance, there is no noteworthy barrier in using FaceCloak, unless we take into account the future possibility to pay for a third party server. However, users have to email keys that are used by the system for decryption. With flyByNight though, users always have to use this Facebook application to view the decrypted text of their private messages. The same is true for Persona. Similarly, NOYB needs to be used not only by the user herself, but also by other users, in order to create a richer public dictionary, hence having more possibilities to substitute the atoms of her attributes. Finally, SIGs solution requires the user to browse through a proxy. On the contrary, with our solution the user does not depend on whether her friends choose to use FaceVPSN. However, the user still has to send by email to her friends the XML files with her details—and it is up to her

friends to decide if they want to install the FaceVPSN extension and view her real details. Differently from our prototype implementation [Conti et al. 2011], the current architecture and implementation transparently handles also the updates of XML profiles. Hence, once FaceVPSN is installed and the initial XML is exchanged, FaceVPSN does not need other pre-requisite to work.

Finally, and most significantly, we reviewed each solution to see if it addresses the problem of "publicly available information" (e.g. does it hide it from Facebook and third party applications?). This comparison is in the last column of Table II, which indicates if the corresponding solution in the row is able to hide the information that Facebook classifies as public (Name and Profile Picture), no matter the privacy configuration set by the user. FaceCloak, flyByNight and NOYB hide only partially those information. In particular, FaceCloak and NOYB are able to hide only text data and not pictures, while flyByNight can hide only message data. Persona does not handle profile information hiding at all. In contrast, FaceVPSN allows users to hide their real Name and Profile Picture.

7.3. Experimental Evaluation

To illustrate the actual work of FaceVPSN, we used a test Facebook account ("Vuamst Van Univ"), and created an XML file with the corresponding pseudo details and real details. The published profile resulting from this test account with pseudo information is shown in Figure 11(a). Having FaceVPSN working, and the proper XML file configured, we observed the proper replacement of information in news feeds, pop-ups, and profile pages. The resulting profile page shown to the user with FaceVPSN in action can be seen in Figure 11(b). Besides the profile page, other elements of Facebook are also handled by FaceVPSN. In [Conti et al. 2011], or just trying out FaceVPSN, it is possible to observe replacement in a chat window, notification window, pop-up window, live news feed, and search bar.



(a) Before using FaceVPSN (b) After using FaceVPSN

Fig. 11. FaceVPSN in action. Profile Page of the test account.

Unfortunately, FaceVPSN (and privacy in general) comes at a price. To be more specific, the usage of FaceVPSN would deprive a user from some of the functionalities of the Facebook platform. A user appearing with a pseudo name in Facebook might not get invited to particular events. Thus, if users are concerned that their real friends will not be able to find them, they can also choose to display an alternate name in addition to the pseudo one.

In our running of FaceVPSN system, we observed that the main issues of FaceVPSN are: i) the time overhead for replacing the pseudo information with real one; and ii) the automatic propagation of an updated XML profile (of User 1), to the friends that are using that XML to see the profile of User 1.

We recall that the Replacer (FaceVPSN component that is responsible of the replacement of the real information) and ProfileUpdater (the component that is responsible for managing the updat-

ing of the XML profile) work independently. Hence, the resulting time overhead of FaceVPSN is motivated as shown in Figure 12.

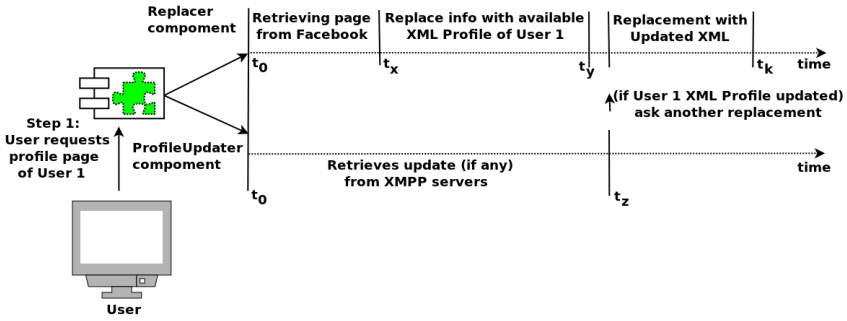


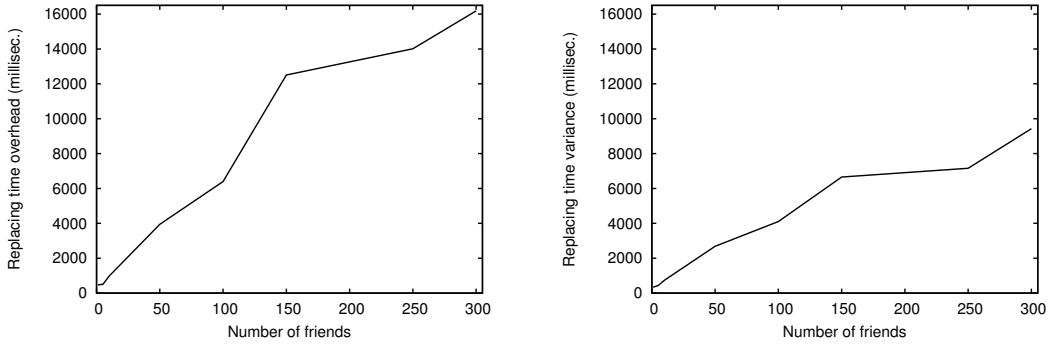
Fig. 12. Generic timeline of FaceVPSN actions.

In particular, while the Replacer is waiting the requested page of User 1 to be retrieved from Facebook, the ProfileUpdater can check if an updated XML profile for the User 1 is available. These actions ends at times t_x and t_z, respectively (see Figure 12). Once the page has been retrieved the Replacer replaces the pseudo info with the real one, this action ending at time t_y. In case the ProfileUpdater actually receives a new update for the profile of User 1, it notifies (at time t_z) the Replacer that an updated XML is available. In case the ProfileUpdater already started the first replacement, or even finished it (as the case shown in the figure), it will start a new replacement considering the updated XML. However, we do not know in advance whether $t_z > t_y$, as well as whether $t_z > t_x$. In fact, if $t_z < t_x$, the ProfileUpdater will not add any time overhead to the browsing experience, on top of the one already given by the Replacer. To conclude, the time overhead observed by the user before getting the facebook page with the replacement of real info for User 1 will be: t_k in the case a new XML has been found during this browsing (and the XML is received once the page is already loaded), t_y otherwise. We expect this latter case to be more probable if User 1 does not update her profile with high frequency. In the following, we investigate the actual value of t_y and t_z, that is the time overhead observed by the user while browsing a page.

We first ran a thorough set of experiments to investigate how the Replacing time overhead varies with the number of friends (we used the regular expressions identified for the proper replacement in all places, and shown in tables I). For evaluation purposes, we used JavaScript Debugger (Venkman [Mozilla 2012d]) of Mozilla to get the profiling data of execution time of various functions of FaceVPSN. The experiments have been performed on a laptop with 2.0GHz Intel i7 CPU, running Ubuntu 12.10. The test have been done with the browser Mozilla Firefox version 14.0.1. to conduct evaluation tests, during which Firefox was ran with FaceVPSN and Venkman extensions. Besides Firefox, no other user applications were running.

We considered a browsing sequence path (e.g. browsing the Facebook home page, then the user profile page, and so on) that includes different types of pages. We used the browsing path proposed in [Conti et al. 2011] (Table IV of [Conti et al. 2011]). In this experiment we considered at the same time: (i) the regular expressions in table I, and (ii) all the replacements running automatically (Table IV of [Conti et al. 2011]). We recall that FaceVPSN does not require that all facebook friends of a profile are using FaceVPSN (indeed, a profile can have only some of its friends using FaceVPSN—which we believe is the realistic scenario). However, for our experiments we conservatively assumed all the friends of a given profile are using FaceVPSN, and for all of them requiring the replacement of all the information handled by FaceVPSN (including pictures). We measured the time overhead for the given browsing sequence path, for 1, 5, 10, 50, 100, 150, 250, and 300 friends. For each number of friends, we run the browsing sequence 10 times, and generated the sample reports with

data about execution times of FaceVPSN functions. The measured values represent the time when all the replacements are performed on a page, rather than how long afterwards will the first replacements be seen. The results obtained from this experiment are summarized in Figure 13(a) and Figure 13(b), for average and variance values, respectively.



(a) Time overhead (average).

(b) Time overhead (variance).

Fig. 13. Replacing time overhead of FaceVPSN.

From Figure 13(a) we observe that, as expected, increasing the number of friends (whose information need to be replaced), there is an increase in the execution time. For example, the average replacing time overhead, is 503 milliseconds for 5 friends in the user's VPSN, 3942 milliseconds for 50 friends, and 6396 for 100 friends in the user's VPSN. This behaviour is understandable when considering the fact that replacements are performed many times for the various friends in the news feed, and also when HTTP requests are generated or URL changes in the address bar. We recall that FaceVPSN performs name replacement in various links (e.g. Friend's First and Last Name in profile page posts, someone's Photos, someone's Links etc.) through regular expressions. In general, we observe that the overhead increases quite linearly with the number of friends. However, we cannot expect that different friends add the same overhead. In fact, the more a friend is active (e.g. posting messages) the more overhead will be added. This characteristic can be noticed comparing the slope of the curves (i) between 100 and 150 friends and (ii) between 150 and 250 friends.

We underline that the times reported here are the ones required for the all replacements in the page to be completed, while the replacement starts just after the page is loaded, i.e. the users can immediately see some of the real information replaced. Also, since the replacement is done in a sequential order, it is highly probable that even the replacement that actually requires more time to be completed (e.g. when browsing the "wall"), in practice is barely noticeable by the user that start reading a couple of posts on top of the page.

We remind that our intention is to provide a solution that, in terms of available features, is as much as possible transparent when compared to the standard behaviour of Facebook. That is we also consider features that are very time demanding like the one with Ajax requests while user is typing a name in a search bar. For example, let us assume the user wants to search for a friend with real name "Name". After user types just "N", a reply (for the Ajax request) with a suggestions list will show the list of all the friends that have a name starting with "N". After the user types "Na" the list will show all friends' names starting with "Na", and so on. While we implemented these features too (that implies continuous replacements), we argue that disabling these features (or having the replacement only upon explicit user-request, e.g. by pressing `F8` as proposed in [Conti et al. 2011]) might further reduce the replacing time overhead, that we consider to be already acceptable.

Based on our experience with FaceVPSN, when a user navigates to some page in Facebook, the first replacements can be seen in less than one second (on average) after the new page is shown.

This varies according to what the user is viewing. In case of a profile page with only a few wall posts, then the replacements are performed in a few seconds. Conversely, if the user is viewing the home page of Facebook, and there happens to be a lot of news feeds with friends, whose information needs to be replaced, then it takes more time.

After evaluating the Replacer time overhead (that is t_y in Figure 12), we evaluated the XML updater component. We remind that in general, the Replacer logical component operates in these cases:

— i) at installation time (to create the N accounts on the N XMPP servers);
— ii) when the user profile is modified (e.g. say the user modifies her real profile picture) the updated XML has to be sent to the interested friends. We remind that the user might use different XML files (i.e. different views—see Section 3) in different VPSNs;
— iii) on the friends' side, the ProfileUpdater has to receive the updated XML from the XMPP Pub/Sub servers.

In the following, we discuss the overhead in these three cases. We observe that: overhead in point i) is experienced only once (installation time); ii) is experienced only when the user updates its profile; finally iii) is experienced each time a page is browsed, and an updated XML for the browsed page is available on XMPP servers—this latter one influencing t_k of Figure 12, hence potentially the browsing experience (only in the case $t_z > t_x$ holds in the specific browsing scenario).

For our experiments, we considered N=10 XMPP servers. In particular, accordingly to the requirements described in Section 6.1, we considered servers that do not require any user interaction (e.g. resolving CAPTCHAs) to create accounts, publish and get messages. Also the selected servers can handle messages for off-line users. In particular, we considered the 10 servers listed on the x-axis of Figure 14(a).

We first investigated the time required during the set-up phase. In particular, we run the script responsible for the set-up of the XML profile updated. We remind that during this phase, FaceVPSN is responsible to create for the user that is installing FaceVPSN, an account for each of the N XMPP servers available. We first measured how much time is required to automatically create an account on each of the considered servers. For each server we connected to the server, created an account, get the confirmation from the server that the account was created, and disconnected from the server. We repeated this operations 100 times (for each server). The experiments have been done from a computer lab of Vrije Universiteit Amsterdam (starting at 5pm of March 21). The resulting average values are shown in Figure 14(a). We expect the values shown to be influenced by the location of the computer used for browsing and the location of the servers (e.g. contacting a server in Japan from Netherlands takes more than contacting a server in Germany). The browsing time might also have an influence on these results. For all servers, we observe from Figure 14(a) that the average time for creating an account is always less than 4 seconds, and less than 2 seconds for 50% of the servers.

Then, we investigated how much time the overall set-up script needs. That is how much time is required, during the installation of FaceVPSN to create all the N accounts on the N different servers. Hence, we measured the time to run all the script (no intermediate measurement were taken). We ran 100 tests, and we obtained as a result an average time of 19067.88 milliseconds (with a standard deviation of 707.70). From these values we conclude that a sequential creation of the required accounts (N=20 in our experiments) could take less than one minute in most cases. This time can be further decreased if different threads take care of the creation of the different accounts. In any case, this operation is performed only once—during the installation of FaceVPSN—and we argue that such a time overhead is highly acceptable. Finally, this operation can also be done in background while other threads take care of other issues of the installation, or when the user if filling the form with her real information to produce the XML.

After investigating the set-up time overhead, we focused on more frequent operations: 1) publishing data when a profile is modified; 2) checking if a new profile is available for the browsed profile and (if so) receive it. We run these experiments considering an increasing number of $n << N$ servers actually used to publish a specific XML profile. For both, publisher and subscriber side,

we measured separately the overhead due to file manipulation, and the overhead due to the actual transmission of the data over the network. Also, in both cases we considered two possible XML profile size, that are 697 and 1575 bytes (in the following also referred as "short" XML, and "long" XML, respectively). The file manipulation includes the creation of the file, the readings and writings, and most importantly the encryption (for the publish case) and decryption (for the receive case). In particular, we used for the crypto operations the AES Rijndael with keys of 256 bits.

The time overhead for publishing an updated XML is shown in Figure 14(b). From this figure we note that, as expected, the overhead increases when the XML file size increases, e.g. considering n=3 servers, the observed average overhead is 3762.88 ms (standard deviation 47.59 ms) for the small XML, while it is 3687.55 ms (standard deviation 58.33 ms) for the long XML. The overhead due to the file manipulation (encryption included) and the one due to the network transmission are 0.522 ms (standard deviation 0.002 ms) and 3762.36 ms (standard deviation 47.60 ms) respectively for the short XML. For the long XML the numbers for the overhead due to the file manipulation and network transmission are 0.546 ms (standard deviation 0.003 ms) and 3687.00 ms (standard deviation 58.32 ms) respectively. Finally, as expected we also observe that increasing the number of servers to be considered, also bring an increase in the overhead in publishing the updated XML.

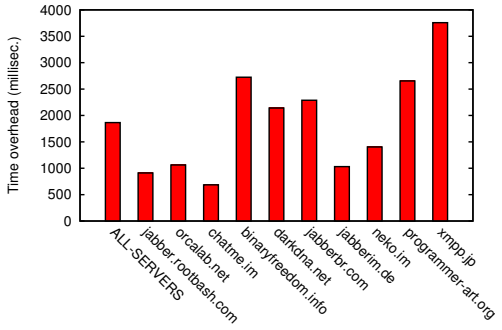
As for the overhead for receiving a XML profile, we considered n=10 (n is the actual number of servers used for transmitting the data, out of the N servers available), while considering all the possible scenarios of servers' availability. We remember that the choice of using not just one server but in general $n \geq 1$ is because we want to provide a solution that is failure and attack resilient. That is, if a number of servers $n' < n$ is down, malicious, or under failure, this does not avoid the propagation of the updated XML. Hence, in the experiment we considered a variable number of servers that does not provide any XML file during the retrieving phase. The results are shown in Figure 14(c), where the x-axis indicate the n-th server at which we were able to correctly receive the XML profile, while the y-axis reports the corresponding time overhead. For example n=10 in the x-axis indicates that the ProfileUpdater contacted the first 9 servers without being able to obtain the XML, and a successful XML has been received only from the 10-th server. We argue that, in practical cases, contacting two or three servers is enough to correctly receive the updated XML.

We remind that the values shown in Figure 14(c) corresponds to t_z in Figure 12. We tested with [AOL 2011] the retrieving time of a Facebook profile page[2] (i.e. t_x in Figure 12), from Amsterdam, considering a DSL connection, and Internet Explorer 7. Out of 10 tests, we get an average of 2.865 seconds to have the page fully loaded. Hence, if we assume being able to receive the XML profile after contacting three servers (i.e. from Figure 12, t_z=1820ms), since t_z would be smaller than t_x, the ProfileUpdater features does not add any time overhead. We conclude by observing that even if the numbers of servers required to be contacted would be a bit more (e.g. three) and even if the download of the page would be smaller, the expected total overhead (in the worst case where an updated profile is actually there to be received) experienced by the user will be some 3 seconds.

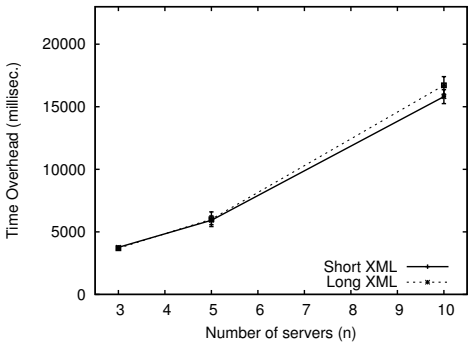
8. CONCLUSION

In this paper, we went through the concept of VPSNs (Virtual Private Social Networks, that we recently proposed in the preliminary version of this paper), via proposing a complete architecture and implementation of the VPSNs concept in the Social Networking Site (SNS) that is by far the most popular SNS: Facebook, counting some 1 billion users. The aim of VPSNs is to mitigate the privacy threats to which personal data of this huge number of users are exposed. FaceVPSN, that is our VPSN implementation for Facebook is, to the best of our knowledge the first solution for privacy threats mitigation that all at once is: lightweight, completely distributed, does not depend on the collaboration from Facebook, does not need to rely on any specific server—i.e. it is also resilient to failure or malicious activities from external servers—, it offers almost the same functionality as Facebook, and is (to some extent) transparent to the user. Implementation and experimental results

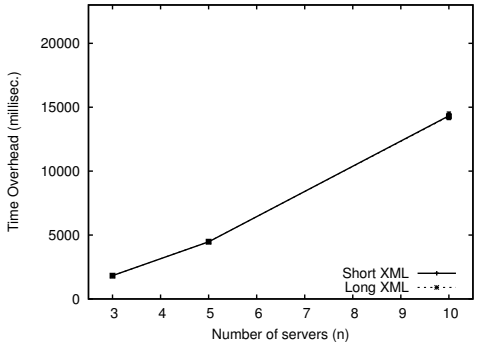
[2]We considered this page: `http://www.facebook.com/#!/profile.php?id=100000523696574&sk=wall`



(a) Account creation time.



(b) Publish XML profile.



(c) Receive XML profile.

Fig. 14. ProfileUpdater Time overhead.

further confirmed the feasibility of the proposal, and shown a very limited time-overhead experienced by the user.

Future research aims to provide solutions to simplify the establishment of a FaceVPSN friendship via interactions of mobile devices (e.g. smartphone), and in turn use these devices as portable storage that interacts with the FaceVPSN component. This would make our proposal independent from the specific computer machine where the user is actually storing the data. Moreover, we intend to leverage new web technologies like Mutation Observers and Web Workers to make the implementation of FaceVPSN (i.e. the Replacer part) more efficient and user friendly. Finally, we aim to investigate privacy threats mitigation solutions for location based services of SNSs, like Facebook Places and Google Latitude.

REFERENCES

AIMEUR, E., GAMBS, S., AND HO, A. 2009. Upp: User privacy policy for social networking sites. In *ICIW '09*. 267–272.

AIMEUR, E., GAMBS, S., AND HO, A. 2010. Towards a privacy-enhanced social networking site. In *ARES '10*. Vol. 0. 172–179.

AOL. 2011. Web page test. `http://www.webpagetest.org/`.

ARCHIVEEXPLOITS. 2012. Facebook's servers was hacked again by inj3ct0r team. `http://inj3ct0r.com/exploits/13403`.

BADEN, R., BENDER, A., SPRING, N., BHATTACHARJEE, B., AND STARIN, D. 2009. Persona: an online social network with user-defined privacy. In *SIGCOMM '09*. 135–146.

BEATO, F., KOHLWEISS, M., AND WOUTERS, K. 2011. Scramble! your social network data. In *PETS '11'*. 211–225.

BOYD, D. M. AND ELLISON, N. B. 2007. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication. 13(1)*, Article 11.

CARMINATI, B., FERRARI, E., MORASCA, S., AND TAIBI, D. 2011. A probability-based approach to modeling the risk of unauthorized propagation of information in on-line social networks. In *ACM CODASPY '11*. 51–62.

CARZANIGA, A., ROSENBLUM, D. S., AND WOLF, A. L. 2001. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems 19,* 3, 332–383.

CONTI, M., HASANI, A., AND CRISPO, B. 2011. Virtual Private Social Networks. In *ACM CODASPY '11*. 39–50.

CUTILLO, L. A., MOLVA, R., AND STRUFE, T. 2009. Safebook: A privacy-preserving online social network leveraging on real-life trust. *IEEE Communicaions Magazine 47(12)*, 94–101.

DANIEL, G., MAXWELL, S., RAPHAEL, S., AND ILYA, Z. 2010. Diaspora*. `http://www.joindiaspora.com/`.

DE CRISTOFARO, E., SORIENTE, C., TSUDIK, G., AND WILLIAMS, A. 2011. Hummingbird: Privacy at the time of twitter. Cryptology ePrint Archive, Report 2011/640. `http://eprint.iacr.org/`.

DURR, M., WERNER, M., AND MAIER, M. 2010. Re-socializing online social networks. *GreenCom & CPSCom '10*, 786–791.

DYBWAD, B. 2010. Facebook and others caught sending user data to advertisers. `http://mashable.com/2010/05/20/facebook-caught-sending-user-data-to-advertisers/`.

FACEBOOK. 2012a. `http://www.facebook.com`.

FACEBOOK. 2012b. Facebook data use policy. `http://www.facebook.com/about/privacy/`.

FELT, A. AND EVANS, D. 2008. Privacy protection for social networking apis. In *W2SP '08*.

FIGUEIREDO, R. J., BOYKIN, P. O., JUSTE, P. S., AND WOLINSKY, D. 2008. Integrating overlay and social networks for seamless p2p networking. In *WETICE '08*. 93–98.

FOUNDATION, X. S. 2012. Xep-0060: Publish-subscribe. `http://xmpp.org/extensions/xep-0060.html`.

GANGULY, A., AGRAWAL, A., BOYKIN, P. O., AND FIGUEIREDO, R. 2006. Ip over p2p: enabling self-configuring virtual ip networks for grid computing. In *IPDPS '06*. 49–49.

GOLBECK, J. 2009. Trust and nuanced profile similarity in online social networks. *ACM Trans. Web 3*, 12:1–12:33.

GROSS, R. AND ACQUISTI, A. 2005. Information revelation and privacy in online social networks. In *WPES '05*. 71–80.

GUHA, S., TANG, K., AND FRANCIS, P. 2008. Noyb: Privacy in online social networks. In *WOSN '08*. 49–54.

HASANI, A. 2012. Virtual private social networks website. `http://sites.google.com/site/fbprivacy2010/`.

HAY, M., MIKLAU, G., JENSEN, D., WEIS, P., AND SRIVASTAVA, S. 2007. Anonymizing social networks. Tech. Rep. 07-19, University of Massachusetts Amherst. March.

ISODE.COM. 2012. M-link server. `http://www.isode.com/products/m-link.html`.

JABBERES.ORG. 2011. `http://www.jabberes.org/servers/`.

JIN, L., TAKABI, H., AND JOSHI, J. B. 2011. Towards active detection of identity clone attacks on online social networks. In *ACM CODASPY '11*. ACM, 27–38.

KACIMI, M., ORTOLANI, S., AND CRISPO, B. 2009. Anonymous opinion exchange over untrusted social networks. In *SNS '09*. 26–32.

KOROLOVA, A., MOTWANI, R., NABAR, S. U., AND XU, Y. 2008. Link privacy in social networks. In *CIKM '08*. 289–298.

KUMARI, P., PRETSCHNER, A., PESCHLA, J., AND KUHN, J.-M. 2011. Distributed data usage control for web applications: a social network implementation. In *ACM CODASPY '11*. 85–96.

LUCAS, M. M. AND BORISOV, N. 2008. Flybynight: Mitigating the privacy risks of social networking. In *WPES '08*. 1–8.

LUO, W., XIE, Q., AND HENGARTNER, U. 2009. Facecloak: An architecture for user privacy on social networking sites. In *CSE '09*. 26–33.

MCCUNE, J. M., PERRIG, A., AND REITER, M. K. 2005. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *S&P '05*. 110–124.

MISLOVE, A., VISWANATH, B., GUMMADI, K. P., AND DRUSCHEL, P. 2010. You are who you know: Inferring user profiles in online social networks. In *WSDM '10*. 251–260.

MOZILLA. 2012a. Chrome registration. `https://developer.mozilla.org/en-US/docs/Chrome_Registration#contentaccessible`.

MOZILLA. 2012b. Http requests observers. `https://developer.mozilla.org/en-US/docs/Setting_HTTP_request_headers`.

MOZILLA. 2012c. Observer notifications. `https://developer.mozilla.org/en/Observer_Notifications`.

MOZILLA. 2012d. Venkman javascript debugger project page. `https://developer.mozilla.org/en-US/docs/Venkman`.

MOZILLA. 2012e. Xpcom nsiprocess interface. `https://developer.mozilla.org/en-US/docs/Code_snippets/Running_applications`.

NARAYANAN, A. AND SHMATIKOV, V. 2009. De-anonymizing social networks. In *S&P '09*. 173–187.

P. MILLARD, P. SAINT-ANDRE, R. M. 2010. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence.

PACI, F., MECELLA, M., OUZZANI, M., AND BERTINO, E. 2011. Acconv – an access control model for conversational web services. *ACM Trans. Web 5*, 13:1–13:33.

PRIMELIFE. 2011. Clique. `http://clique.primelife.eu/`.

REAY, I., DICK, S., AND MILLER, J. 2009. A large-scale empirical study of p3p privacy policies: Stated actions vs. legal obligations. *ACM Trans. Web 3*, 6:1–6:34.

ROWSTRON, A., KERMARREC, A.-M., CASTRO, M., AND DRUSCHEL, P. 2001. Scribe: The design of a large-scale event notification infrastructure. In *In Networked Group Communication*. 30–43.

SAINT-ANDRE, P. 2011a. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120.

SAINT-ANDRE, P. 2011b. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 6121.

SORNIOTTI, A. AND MOLVA, R. 2010. Secret interest groups (sigs) in social networks with an implementation on facebook. In *SAC '10*. 621–628.

TABAKOFF, N. 2009. Facebook users are sitting ducks for identity theft. `http://www.dailytelegraph.com.au/news/facebook-users-sitting-ducks-for-identity-theft/story-e6freuy9-122580713389/`.

THECOCCINELLA.ORG. 2011. `http://thecoccinella.org/servers/servers_by_pubsub_pep.html`.

VAN AMSTEL, B., GROENEVELD, F., AND BORSBOOM, B. 2010. Please rob me. `http://pleaserobme.com/`.

VU, L.-H., ABERER, K., BUCHEGGER, S., AND DATTA, A. 2009. Enabling secure secret sharing in distributed online social networks. In *ACSAC '09*. 419–428.

WOLFE-WYLIE, W. 2010. The harm of facebook pictures. `http://www.torontosun.com/life/2010/08/10/14978476.html`.

XMPP PROTOCOL. 2011. `http://xmpp.org//`.

XMPP.ORG. 2011. `http://xmpp.org/services/`.

YOUNG, A. L. AND QUAN-HAASE, A. 2009. Information revelation and internet privacy concerns on social network sites: a case study of facebook. In *C&T '09*. 265–274.

YUKSEL, A. S., YUKSEL, M. E., AND ZAIM, A. H. 2010. An approach for protecting privacy on social networks. *ICSNC '10*, 154–159.

ZHELEVA, E. AND GETOOR, L. 2009. To join or not to join: The illusion of privacy in social networks with mixed public and private user profiles. In *WWW '09*. 531–540.