BlogCrypt: Private Content Publishing on the Web

Tamás Paulik
Dept. of Telecommunications
Budapest University of Technology and Economics
Budapest, Hungary
paulik.tamas.email@gmail.com

Ádám Máté Földes
Dept. of Telecommunications
Budapest University of Technology and Economics
Budapest, Hungary
foldesa@hit.bme.hu

Gábor György Gulyás
Dept. of Telecommunications
Budapest University of Technology and Economics
Budapest, Hungary
gulyasg@hit.bme.hu

Abstract—**Voluntary disclosure of personal information is becoming more and more widespread with the advent of Web 2.0 services. Publishing such information constitutes new kinds of threats, such as further reinforcing already existing profiling techniques through correlation of perceived user activities to those publicly disclosed, but the most obvious of all is the intrinsic threat that malicious third parties collect and combine information we publish about ourselves. In this paper, we evaluate currently existing solutions that are destined for addressing this issue, then propose a model of our own for providing access control for a user over information she published and analyse our implementation thereof.**

Keywords-profiling; web privacy; user content; Web 2.0

I. Introduction

'Profiling' is a term expressing the tracking of the activity of users or visitors among certain services, e.g. a website, for various purposes – the most probable motivation being financial benefit through targeted advertising [4]. While users of a service can, in some cases, certainly benefit from the personalised content found on websites (e.g. in the form of articles of their interest in an online newspaper), and research shows that actual behaviour does not always match what users claim about their objections against tracking when certain benefits are offered in turn [5], profiling is a privacy concern for many users. For example, it was shown in a survey over German Internet users that 60% had avoided a website in order to protect their privacy, and it is safe to state that a high proportion of users have serious doubts about efficiency and the integrity of the data protection measures of commercial websites [5]. It must also be noted that the technology has evolved since then, which poses even bigger problems about privacy of online activities, as shown later in this paper.

The 'classical' method of profiling through third party cookies [2], further reinforced by the use of Flash cookies (i.e. local stored objects or LSOs of Adobe Flash Player), is living its renaissance [6]. A cookie is a file that can be planted on the user's hard drive for later retrieval by a website [4]. Cookies have many purposes, one of which is making up for the connectionless nature of the HTTP protocol by identifying a user even after the end of the HTTP request. A service that requires authentication (e.g. a webmail server) usually relies on so-called session cookies to recognise that certain HTTP requests belong to the same user. However, third parties, e.g. advertisers, can also install a cookie containing an identifier, and fetch it on every website that embeds their banners, thereby tracking the user between requests made to different services. In contrast to HTTP cookies that privacy-aware users reject or delete at regular intervals, thereby making the advertiser overestimate the number of unique visitors, Flash cookies tend to 'survive' since many users do not even know that they exist, and they are not affected by the 'Private Browsing mode' implemented in modern browsers like Internet Explorer 8 and Firefox 3 [6]. An advertiser can even recreate HTTP cookies from Flash cookies through a Flash animation embedded into a third party website.

The appearance of Web 2.0 has also led to the evolution of different approaches to profiling. For instance, a service provider can offer various services to the user (as is the case with Gmail, Google Calendar, Google Groups, Picasa etc.), each of which logs activity to some extent. These logs can be combined, and the provider gets a detailed image about the behaviour of the user in many cases. It must be noted that this kind of tracking does not mandate using sophisticated techniques like Flash cookies or client-side profiling software – it is effectively the user who voluntarily profiles herself for the service provider.

A service provider can also use information gathered from other websites. For example, a social networking website can retrieve the missing information in a user's profile by using the information found on the user's blog at another service provider, as was the case with Facebook until a recent change in their privacy policy [7].

It must be noted that there exist privacy enhancing extensions to profiling, e.g. profiling that is performed on perturbed or obfuscated data [8]. However, a privacy-aware user may be concerned that such data alterations are not performed before the information is sent to the central server.

All this discussion leads us to the conclusion that users cannot entrust service providers with the protection of their privacy. However, service providers are not the only threat to online privacy. Anybody can crawl the public sources of information on the Web in an attempt to find information about a target. Using pseudonyms is not necessarily sufficient for concealing our identity, as shown in Section II.

All this discussion supports the motivation that a user must be able to control the access to the information she publishes about herself. Our main contribution in this paper is the proposal of such a privacy-enhancing tool that enables fine-grained disclosure of private data. Our approach intends to be general and does not target a single service or a single type of service (e.g. social networking sites). Furthermore, it does away with the use of trusted third parties, delegating the responsibility of the protection of the information entirely to a client-side application.

In Section II we discuss the threats arising from profiling using publicly available data and compare the already existing solutions for this problem. Section III describes the requirements we stipulated for a practical solution, followed by the discussion of our proposed model in Section IV. In Section V, we analyse our implementation of the aforementioned model. Finally, in Section VI, we conclude and discuss the future of our work.

II. Profiling by using public information available on the web

There certainly are situations where the user can also benefit from being profiled. For instance, there are users who claim that their web browsing experience is enhanced by targeted advertising, since they only see advertisements about goods that they are interested in, while others say that not being bombarded by the same banners upon each download of a webpage is better than having to view the same advertisement all the time [4]. However, profiling can also constitute a threat to privacy, as described below.

Narayanan and Shmatikov describe an attack where two social network graphs can be correlated to make a mapping between nodes that denote the same user [3]. The re-identification has a stunningly high success rate of 31%. If we suppose that an attacker can find the personally identifying information for a node of one of these networks, she can enrich the user's profile by complementing the already obtained information from the other one. The privacy of such a user, who published information on the second social network under the belief that her pseudonym is sufficient to protect her anonymity, is then clearly violated.

Diaz, Troncoso, and Serjantov discuss an attack where the adversary, who is in the possession of a social network graph, infers the communicating parties in a mix network [9]. The paper makes the assumption that interaction between the users is well described by how they exchange private messages on the social network. Under these circumstances, it has been shown that sender and recipient anonymity of the mix is compromised, i.e. the effective anonymity set is reduced to its fifth, and anonymity does not increase when the social network grows.

These novel attacks lead us to the conclusion that publicly available information, e.g. which is published on social networks can be used efficiently to match user activity between different services, regardless of the pseudonyms the user may choose. To mitigate the risks arising from malicious third parties using publicly disclosed information, there is a need for solutions to address this issue. Therefore, after evaluating currently existing implementations, we suggest a model where the disclosure of information can be limited to a set of users, and analyse our own implementation thereof.

III. Analysis of current solutions, requirements

In this section, we propose requirements against a practical solution. Then, in the second part of the section, we discuss the merits and drawbacks of the currently existing implementations, based on a categorisation scheme that we designed.

A. Requirements against a practical solution

We have defined the following requirements against a solution that we consider both practical and versatile.

Gradual deployability. Users of a service are likely to prefer to continue providing public information for those who do not install the software. If the software renders the service useless for everyone but those who have it installed, users are less likely to adopt the solution.

Full autonomy. An average computer user is unlikely to be willing to hassle with the installation of third party software like external cryptographic libraries. Therefore, we prefer containing the entire software in a single installation package.

Furthermore, service providers may choose to take a different approach to the privacy enhancing solution as time passes, e.g. through taking down a trusted server, blocking the distribution and operation of the application through their API, or even implementing man-in-the-middle attacks by faking a trusted third party. We find it easier to guarantee the security of the solution by delegating the responsibility of securing its environment entirely to the user.

Universality. Users may find it more practical to use a tool that is compatible with many services. Furthermore, should the service provider decide to take countermeasures against the privacy enhancing software, users can still continue to use it on other sites. Although we think that the ideal solution operates on a universal interface, the general interface of a textfield is sufficient, since most services use this as the means of publishing information.

Usability without compromises and easy installation. The user is likely to quickly stop using a solution that seriously harms the usability of the service or that is cumbersome to install or configure.

Providing strong confidentiality. The privacy enhancing software must be able to guarantee the confidentiality of the information against third-parties, i.e. through means of strong cryptography. We aimed for the use of symmetric key cryptography for the sake of simplicity, although the implementation of a more complex solution using asymmetric key cryptography may be desirable.

No protection against discovery. While we think that a solution that uses both steganography and cryptography is the ideal implementation, the former is hard to implement, mainly due to the imposed capacity limitations. However, considering that discoverability can make certain service providers hinder the use of the software, an option to use steganography could be a considerable improvement in the future. It must be noted that certain solutions we analysed used fake data to substitute the ciphertext, thereby concealing the act of ciphering. We prefer steganography over fake data, because the latter makes the solution service specific.

B. Evaluation of current implementations

In order for us to be able to define the requirements against our solution, we devised a categorisation scheme which aids us in classifying the currently existing solutions (referred to as 'software' below), point out their deficiencies, and evaluate our design. We have made our evaluation regarding the following attributes:

1. Gradual deployment
   a. Possible: The service is not rendered useless if not all users adopt the software.
   b. Not possible: All users must adopt the software if the usability of the service is to be maintained.
2. Autonomy
   a. Completely autonomous (Full): The software uses no resources that are not under its control.
   b. Third party server needed (TPSe): The software requires a server that is operated by a trusted third party.
   c. Third party software needed (TPSo): The operation of the software is supported by other software developed by a third party.
   d. Service provider collaboration needed (SPCN): The service over which the software is to be operated must cooperate to maintain the usability of the software.
3. Universality
   a. Service specific (SS): The software can only be used over a set of well-defined services.
   b. Uses general interface (GI): The software is operated over a well-defined interface that is used by many services. An example of such an interface is a textfield.
   c. Uses universal interface (UI): The software uses an interface that all services have; therefore the software can cooperate with all services. An example of such a solution is an intelligent agent that makes sure that all information the user provides reaches the service provider in an encrypted form.
4. Compromises
   a. All functions remain usable (100%): The software does not hinder the use of the service at all.
   b. The majority of the functions of the service remain usable (51%-99%)
   c. Some functions remain usable (25%-50%).
   d. The service is rendered useless (0%-24%).
5. Comfort of use
   a. The software requires minimal interaction, e.g. selecting content and keys (Min)
   b. The software requires more interaction than minimal (Normal)
   c. The use of the software is uncomfortable (Uncomf)
6. Comfort of installation
   a. No configuration needed (NoConf)
   b. Some configuration is needed, but it is more or less straightforward (Min)
   c. Installation and configuration takes long, and requires expertise (Compl)
7. Encryption
   a. Symmetric cipher (Sym)
   b. Asymmetric cipher (Asym)
   c. Both types of cipher (Both)
   d. Custom algorithm (Custom)
8. Discoverability
   a. The software uses strong cryptography, but its presence is easily discovered (Crypto)
   b. The software uses strong cryptography and replaces encrypted information with fake data which appear to be real (Crypto+Fake)
   c. The software uses steganography (Stego)
   d. The software uses steganography and cryptography (Stego+Crypto)

We have summarised our evaluation of the currently existing solutions in Table I.

TABLE I. CLASSIFICATION OF CURRENT SOLUTIONS

Program	**Gradual Deployment**	**Autonomy**	**Universality**	**Compromises**	**Comfort of Usage**	**Comfort of Installation**	**Encryption**	**Discoverability**
Lockr [11]	Possible	TPSe, TPSo, SPCN	SS	100%	Min	N/A	Both	Crypto
FaceCloak [12]	Possible	TPSe	SS	51%-99%	Min	NoConf	Both	Crypto+Fake
NOYB [1]	Possible	TPSe	SS	51%-99%	N/A	N/A	Custom	Crypto+Fake
FlyByNight [10]	Possible	SPCN	SS	51%-99%	Normal	NoConf	Both	Crypto
FireGPG [15]	Possible	TPSo	GI	51%-99%	Normal	Compl	Both	Crypto
Ideal	Possible	Full	UI	100%	Min	NoConf	Both	Stego+Crypto

IV. SUGGESTED MODEL

We suggest a 4-phase model for our implementation, excluding the creation of the content itself (see Fig. 1.):

1. Defining authorised users. Securing the information, i.e. by the means of strong cryptography.
2. Posting the secured information onto the site of the service provider.
3. Manual distribution of keys out-of-band.
4. Revealing the information for authorised users.

A. *The operation of BlogCrypt*

We implemented our solution called BlogCrypt [16] as a FireFox extension. It features a user interface for defining keys, encrypting textual information etc. The user types the information into the interface of the service where she wishes to post, and selects the text (e.g. an entire blogpost or the most sensitive parts of it) to be encrypted. Provided that the text is in a textfield or textbox, BlogCrypt replaces the highlighted information with its encrypted version, encoded in Base64. The encrypted block has a header which contains an identifier for the key, the cipher that is used and the length of the encrypted block. The inspiration for the format of the header was BBCode; the header is effectively a *[crypt]* tag with the attributes *keyid*, *algo* and *length*, but without a closing counterpart, as many services filtered our tag when it was closed by *[/crypt]* in a syntactically correct fashion. As such, the *length* attribute was defined as necessary in order to make up for the missing closing tag.

Our means of defining authorised users is defining cryptographic keys for each of them. BlogCrypt supports a hierarchical authorisation model, which means that a certain piece of information can be secured by multiple keys. The user needs to be in the possession of all keys in the hierarchy in order to be able to decrypt the information. The uniqueness of the key identifiers must be guaranteed within a chosen DNS domain by default; however, the user can choose to make a key identifier globally valid. All keys are always stored locally to the web browser instance.

BlogCrypt secures the information by performing encryption with AES-CTR or AES-CBC. These algorithms, among other useful features such as Base64 encoding, are provided by the PidCrypt library [17], which we include in the installation package. The authorisation hierarchy is realised by multiple encryption of the same block. However, the encrypted block can contain arbitrary information, which means that the decryption of a block can yield an ensemble of plaintext and other encrypted blocks that can be decrypted with a key that is a child of the current key in the hierarchy.

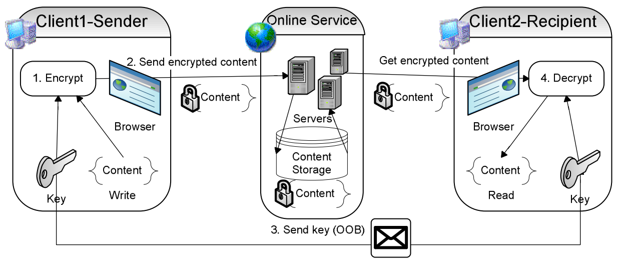


Figure 1. Our suggested model

An encrypted block can be revealed provided that the user is in the possession of the key that was used to encrypt it. BlogCrypt looks for the key identifier of the block within the local key database. As an explanation of the role of the different types of keys, let us consider the example key database of Table II. Suppose that the user visits a webpage under the DNS domain *a.com*, and finds an encrypted block with key identifier *1*. Both domain and global keys exist, but the domain key <*a.com, 1*> is used for decryption, since domain keys are preferred over global keys. If the webpage had contained an encrypted block with key identifier *4* instead of *1*, the global key with identifier *4* would have been used for decryption. It must also be noted that, while key identifier 2 is present under twice in the database, it is unique within each DNS domain. Therefore, if key identifier *2* is found on a downloaded webpage, BlogCrypt chooses <*a.com, 2*> as the decryption key for pages under *a.com*, while <*b.com, 2*> is chosen for webpages hosted under *b.com*. If, for a given encrypted block, no suitable key is found in the database, decryption fails, and the encrypted block is substituted by an error message.

Decryption is automatic in BlogCrypt, provided that the necessary keys are imported into the database. However, there are situations, such as editing one's own blogpost that contains encrypted blocks already, where this is undesirable. BlogCrypt features a function called Editor Mode for this reason. This turns off automatic decryption for the time being. In other cases, such as downloading a webpage with AJAX content, automatic decryption does not work. This is because the browser fires the *onLoad* event when the webpage is loaded. At this point however, the AJAX parts are mere placeholders that are populated later. When all parts of the webpage have finished loading, the user can initiate manual decryption. This causes BlogCrypt to rescan the webpage for encrypted blocks and decrypt each of them again.

B. *Key management*

This section discusses the limitations of the currently implemented key management, and lists the challenges of designing a more advanced scheme.

1) *Current key management possibilities*

Key management is a vital issue in every solution that uses encryption and is destined to share some secret with a larger group of users. The current version of BlogCrypt features only manual importing-exporting of keys. This is not a problem for only a few users (i.e. few keys), but it becomes quite cumbersome when the number of authorised users or groups is increasing. The different aspects of key management (i.e. publishing the content in an encrypted form, deny-

TABLE II. AN EXAMPLE KEY DATABASE

Domain	**KeyId**	**Key**
a.com	1	a
a.com	2	b
a.com	3	c
b.com	2	d
global	1	e
global	4	f

ing access to an authorised user or group, redistribution of keys) can be highlighted in the following situations:

Access is provided to multiple users. A key is created for each user, and then the information is encrypted with each. The encrypted blocks are posted onto the service. This works for a little number of authorised users, because the number of encrypted blocks and the burden of key distribution increase linearly in the number of users. However, denying access to a user means simply deleting the encrypted block corresponding to her key.

Access is provided to multiple disjoint groups of users. This is a generalisation of the previous situation. The information needs to be encrypted with multiple keys, i.e. one for each group, and then each encrypted block needs to be posted onto the service. The order of magnitude of keys is proportional to the number of groups, and – in order of magnitude – inversely proportional in the number of users in the largest group. Denying access to a member of a group means creating a new group key, encrypting the information with it, reposting the encrypted block, and then distributing the keys to the remaining members of the group through an out-of-band mechanism. This means that the burden of key redistribution is proportional in order of magnitude to the number of users in the largest group. Denying access to an entire group is done by deleting the corresponding encrypted block.

Access is provided to multiple groups of users where groups intersect each other. This is the most general situation to be considered. Publishing the information is the same amount of work as in the previous situation. However, since a member may possess multiple group keys, prohibitive key revocation (i.e. one considers that users in a group are not trusted) means that all groups have to have their keys redistributed which intersect with the untrusted group.

These situations are encountered by different types of users. The first one is the casual blogger: she occasionally creates some content (e.g. some news about her last vacation) which she only shares with some of her friends. Her possibilities range from creating a separate key for each friend to forming a big group of all of them. Her choice is likely to depend on whether she opts for the ease of denial of access or that of publishing the information.

The second type of user is the social networker: she intends to share her content with groups from a social network. She signs up for each, and creates a key for each of them. The groups most likely intersect at some users, which means that we are facing the third situation above. However, the user practically cannot distinguish members of a group. As such, prohibitive key revocation is not possible.

The third type of user is the 'double agent': she distributes information to completely different groups of interest. For instance, suppose that a student would like to throw a party for her birthday. She presumes most of her family will come, but she sends an invitation to her friends as well. Since the room in her flat is limited, she retracts the invitation sent to the friends as soon as they are approaching the maximum.

All this considered, we believe that easy denial of access should not be the primary goal of the user, since any authorised viewer may copy the decrypted content to her hard drive and publish it again someplace else if her access is revoked. Rekeying is somewhat more relevant, e.g. when a group key is believed to be compromised, so as to mitigate the posterior impact of the incident. We emphasise that easy publishing is the most important factor for the user.

2) Other issues

It is safe to say that the implementation of some advanced key management mechanism would be a huge improvement in BlogCrypt. We listed some of the ideas we are currently examining in Section V. However, there are some factors that must be considered when designing a key management scheme. We are examining the following issues.

Authentication. Exchanging keys is a vulnerable process. In order to eliminate the possibility of man-in-the-middle attacks, the communicating parties need to authenticate each other. Public key infrastructures (PKIs) are a widely deployed solution for this problem [13], but their management can be cumbersome, if not completely unfeasible. We think that a distributed 'web of trust' like PGP is more suitable in this application than a PKI that is somewhat centralised by nature.

Media for automatic key distribution. Social networks can be used efficiently for the purpose of key distribution. For instance, a Facebook application could be utilised to make the process fully automatic, e.g. through fetching the public key that was posted onto the profile of a user. Since many social networking sites allow users to upload photographs, the steganographic capacity thereof could also be exploited for exchanging keys.

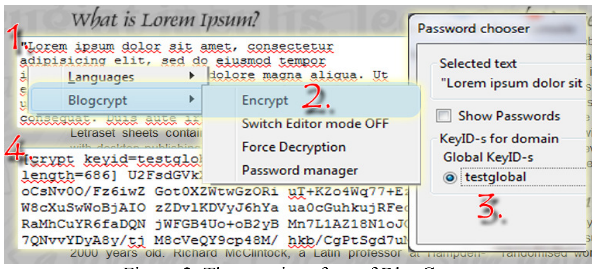
Advanced trust management. Role-based access control could be a way to assign roles to people and provide access based on their memberships. There are schemes that make this concept self-enforcing without central entities [14].

V. BLOGCRYPT: ANALYSIS OF THE IMPLEMENTATION

The chosen platform strictly defined the tools of development. The user interface (see Fig. 2.) was programmed in XUL, while the logic was implemented in JavaScript. These two languages allow easy retrievals from and substitutions into web pages, besides AES encryption with the needed efficiency.

The implementation is divided into separate modules which correspond to the main functionality described earlier. The most important modules, i.e. encryption, decryption and key management, are loosely coupled.

As mentioned earlier, BlogCrypt supports entirely manual key distribution only in its current state. The user may



```
"Lorem ipsum dolor sit amet, consectetur
adipisicing elit, sed do eiusmod tempor
                          dolore magna aliqua. Ut
consequat. Duis aute ir

[crypt keyid=testglo
length=686] U2FsdGVk
oCsNv0O/Fz6iwZ Got0XZWtwGzORi uT+KZo4Wq77+E
W8cXuSwWoBjAIO zZDvlKDVyJ6hYa ua0cGuhkujRFe
RaMhCuYR6faDQN jWFGB4Uo+oB2yB Mn7L1AZ18N1oJ
7QNvvYDyA8y/tj M8cVeQY9cp48M/ hkb/CgPtSgd7u
```

Figure 2. The user interface of BlogCrypt

create, modify, delete, import, export symmetric keys and distribute them out-of-band. The implications of this are discussed in Section IV.

Furthermore, we have analysed our implementation below in regards to the requirements we had stipulated.

Gradual Deployment: Possible. A user protecting her information using BlogCrypt has no effect on the operation of the service at other users.

Autonomy: Completely autonomous. The program does not require additional third-party software, an external server or the cooperation of the service provider.

Universality: Uses general interface. The program uses textfields, which is a common property of the vast majority of today's services.

Compromises: 51%-99%. The operations that require server side applications to interpret the data (e.g. searching) fail. It must be noted that since our goal was to hide the information from third parties including the service provider itself, this is not a fault in our design, rather a feature.

Comfort of Usage: Mostly minimal interaction. Apart from the creation of keys, the operation is fully automatic (except on pages with AJAX content). In the long run, the only user interaction required is to mark the text to be encrypted.

Comfort of Installation: No configuration. The software can be installed with a single click, similarly to other Firefox extensions.

Encryption: Symmetrical. BlogCrypt uses the symmetrical cipher AES.

Discoverability: Cryptography. The use of the software is easily discovered.

Summarising the above, the practical realisation of our theoretical model is ideal according to most of the categories defined in our evaluation scheme, and conforms to the requirements. As such, we consider the implementation as successful as far as our goals are concerned.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have defined a model for access control of published information, classified other solutions for the same task, and provided the details of our reference implementation called BlogCrypt. We believe that while our solution is simple, it not only matches the power of other implementations, but no other evaluated solution delivered all of the key values of BlogCrypt on its own. User-friendliness is another merit that we would like to emphasise.

However, implementing a proper key management scheme could provide a boost to the practical value of the implementation. As such, our further research is likely to focus on the issues of key management and distribution. We are going to examine the possibilities of implementing a digital envelope scheme (i.e. where the message is encrypted with a message key which is in turn encrypted with the keys of the intended recipients, and both the encrypted message and the encrypted keys are posted together) based on public key cryptography. Discoverability can also be an issue with certain service providers and/or countries; we will certainly consider implementing steganography besides cryptography with the aim of making the operation of our software completely impossible to discover by unauthorised third parties. However, we emphasise that such an implementation must retain the key merits of the current version of BlogCrypt, and must not make the solution impractical.

REFERENCES

[1] S. Guha, K. Tang, and P. Francis, "NOYB: privacy in online social networks," Proc. of the first workshop on Online social networks, August 2008., pp. 49–54, doi:10.1145/1397735.1397747.

[2] G. Gulyás, R. Schulcz, and S. Imre, "Comprehensive analysis of web privacy and anonymous web browsers: are next generation services based on collaborative filtering?", Joint SPACE and TIME International Workshops 2008, Trondheim, Norway, June 2008.

[3] A. Narayanan and V. Shmatikov, "De-anonymizing social networks," Proc. 30th IEEE Symposium on Security and Privacy, IEEE Press, March 2009, pp. 173–187, doi:10.1109/SP.2009.22.

[4] R. Pitofsky, S. F. Anthony, M. W. Thompson, O. Swindle, and T. B. Leary. Online profiling: a report to Congress. June 2000.

[5] B. Berendt, O. Günther, and S. Spiekermann. "Privacy in e-commerce: stated preferences vs. actual behavior," Communications of the ACM, vol. 48, issue 4, April 2005, pp. 101–106, doi:10.1145/1053291.1053295.

[6] A. Soltani, S. Canty, Q. Mayo, L. Thomas, and C. J. Hoofnagle, "Flash cookies and privacy," August 2009, Available at SSRN: http://ssrn.com/abstract=1446862.

[7] Privacy policy | Facebook, http://web.archive.org/web/20080719134042/http://www.facebook.com/policy.php, retrieved on January 29th, 2010.

[8] A. Kobsa, "Privacy-enhanced web personalization," P. Brusilovsky, A. Kobsa, and W. Nejdl (Eds.): The Adaptive Web, LNCS 4321, pp. 628–670, 2007. Springer-Verlag Berlin Heidelberg 2007.

[9] C. Diaz, C. Troncoso, and A. Serjantov, "On the impact of social network profiling on anonymity," N. Borisov and I. Goldberg (Eds.): PETS 2008, LNCS 5134, pp. 44–62, 2008.

[10] M. M. Lucas and N. Borisov, "FlyByNight: mitigating the privacy risks of social networking," Proc. of the 7th ACM workshop on Privacy in the electronic society, October 2008, pp. 1–8, doi:10.1145/1456403.1456405.

[11] A. Tootoonchian, K. K. Gollu, S. Saroiu, Y. Ganjali, A. Ganjali, and A. Wolman, "Lockr: social access control for Web 2.0.," Proc. of the first workshop on Online social networks, August 2008, pp. 43–48, doi: 10.1145/1397735.1397746.

[12] W. Luo, Q. Xie, and U. Hengartner, "FaceCloak: an architecture for user privacy on social networking sites," Proc. 2009 International Conference on Computational Science and Engineering, IEEE Press, August 2009, pp. 26–33, doi: 10.1109/CSE.2009.387.

[13] B. Schneier. Applied Cryptography, John Wiley & Sons, Inc., United States of America, 1996.

[14] F. Beato, M. Kohlweiss, and K. Wouters. Enforcing access control in social network sites. Katholieke Universiteit Leuven, Belgium, 2009.

[15] FireGPG – Welcome to the official website of FireGPG!, http://getfiregpg.org/s/home, retrieved on February 19th, 2010.

[16] BlogCrypt, http://pet-portal.eu/blogcrypt/download.html, retrieved on February 20th, 2010.

[17] pidCrypt – pidder's JavaScript crypto library, https://www.pidder.com/pidcrypt/, retrieved on April 10th, 2010