StegoWeb: Towards the Ideal Private Web Content Publishing Tool

Tamás Besenyei, Ádám Máté Földes, Gábor György Gulyás, Sándor Imre
Department of Telecommunications
Budapest University of Technology and Economics
Magyar tudósok körútja 2., 1117 Budapest, Hungary
tamas@besenyei.net, foldesa@hit.bme.hu, gulyasg@hit.bme.hu, imre@hit.bme.hu

Abstract—**Privacy breaches through profiling constitute a considerable threat to users of Web 2.0 services. While many concepts have been proposed to address this issue by allowing users to encrypt, obfuscate, or otherwise conceal information of their choice, all have certain limitations. In this paper, we survey the available solutions, and propose a taxonomy for classifying them based on a revised evaluation scheme that builds upon our previous work. Our main contribution is a model that harnesses steganographic techniques in order to hide sensitive data, and the description of a proof-of-concept implementation thereof that allows a user to hide profile data on a website without installing any sort of software aside from a conventional web browser.**

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

I. INTRODUCTION

As the use of Web 2.0 services – most notably Social Networking Sites or SNSes – is becoming more and more widespread, the privacy-related questions of the sensitive information published there gain significance in a similar tact. The term 'profiling' is used to describe activities which involve collecting data about a person from various sources (e.g. customer preferences in a webshop and personal data published on social networking websites), and merging the pieces of information into a single record, called a profile. Since Web 2.0 services are based on user-created content, profilers can use these services to complement their profiles [4]. Accurate user profiles serve as useful bases for many dubious or outright malicious activities, including targeted advertising and dynamic pricing. This tendency is likely to get worse as real-time searching becomes a core feature in search engines, which makes revocation of information impossible. Therefore, this problem is gaining importance frighteningly fast.

The techniques of profiling have evolved greatly since the birth of the World Wide Web. When IP addresses were fixed, they could be used to identify a user on the Web. Later on, as Internet Service Providers adopted the use of dynamic IP addresses, the main basis of identifying a user became unique identifiers in HTTP cookies and, later, 'Flash cookies' or LSOs [2]. The evolution of tracking techniques is continuous; the concept of Evercookies [15] and the Panopticlick browser fingerprinting experiment [16] indicate that research and improvements in the area have certainly not concluded. Furthermore, information superpowers – service providers that offer a wide range of products to their users – are a major threat [4], because they can have access to various data about the user.

As such, there is a need for applications that protect the user against these actors through limiting the information of personal nature that a profiler potentially has access to. Our previous work [4] introduced such a piece of software called BlogCrypt, a Firefox extension that could encrypt and decrypt data on websites with as little user interaction as possible. In that paper, we showed that BlogCrypt was an efficient countermeasure against profiling, but, as it does not conceal encryption, users are likely to face countermeasures on Web 2.0 sites where encrypting or otherwise obfuscating user content is forbidden by the Terms of Use.

Our main contribution in this paper is a steganographic approach to this problem, which, albeit not a direct successor or an improved version of BlogCrypt, addresses the same issue as it did, but in a slightly different context. The main reason is that steganography is 'expensive', i.e. only a small amount of data can be stored with such techniques. Therefore, while BlogCrypt was a useful solution to encrypt blog posts, StegoWeb is more likely to be applicable in the context of profile data on SNSes. If our application is used for this purpose, a profiler will not be able to link our personal information to other data she has potentially obtained about us.

The paper is structured as follows. In Section II, we survey already existing implementations and concepts that are destined to hinder profiling, and provide a taxonomy for classifying them. Then, in Section III, we discuss our own implementation, and analyse it in terms of advantages and drawbacks. In Section IV, we evaluate our implementation from the aspect of key management, and propose some improvements. Section V describes how the concept can be used for identity management purposes. Finally, we conclude our work in Section VI.

II. EVALUATION OF EXISTING SOLUTIONS

In this section, we discuss the already existing solutions for the aforementioned issues, and categorise them into a taxonomy. Some of these solutions are discussed and classified in our previous work [4].

A. Existing Solutions

There are many different solutions for protecting user content on Web 2.0 sites. We discussed the merits and

shortcomings of most of them in our previous work [4]. The currently available solutions can be categorised as follows:

1) Universal applications. These applications can cooperate with arbitrary services that provide a generic interface such as a textbox. Some of them (e.g. BlogCrypt and NOYB – Secret Messaging [9]) are implemented as standalone Firefox extensions, while others need additional external software to operate (e.g. FireGPG [10]).

2) Site-based applications. These programs (or models) are destined to work with a single Web 2.0 website. Examples of such applications include Lockr [11], FaceCloak [12] and FlyByNight [13]. Newer concepts include Persona [7], which is essentially a privacy-enhanced SNS. As a proof-of-concept implementation, the authors integrated their model with Facebook in such a way that the backend is a Facebook application. Another site-based application is FaceVPSN [8], which is a Firefox extension that allows users to import fellow Facebook users' profile data into a local database, and substitute the corresponding attributes with the locally stored information (if available) when the profile page of a user is loaded. Lastly, SeGoDoc [6] is essentially a middleware for encrypted storage on Google Docs. It is implemented as a Firefox extension, and works on-the-fly, completely automatically.

3) Models without available implementations. These are models that were published in academic research, but their implementation is not available. Examples for this are NOYB – Social Networking [14] and an unnamed community-based access control concept that – similarly to BlogCrypt – assumes server-side storage of encrypted data [5].

B. Evaluation Model

In our previous work [4], we discussed a categorisation scheme for the available solutions, and described the principles based on which its attributes are defined. The revised version of the categorisation scheme is summarised in Table I. We do not discuss the results that have already been published in our previous work, and listed only some solutions that have appeared since then, namely SeGoDoc, Persona, FaceVPSN and our own solution StegoWeb. Furthermore, we have done away with the attribute 'Autonomy', and defined new attributes and categories, too:

1) Key distribution. Possible categories: manual (M), partially automatised (PA), fully automatic (Auto).

2) Independence. Possible categories: operating system independent (OS), browser independent (B), service independent (S).

3) Realisation. Possible categories: external software (ExSw), browser extension (Ext), bookmarklet (B).

C. Taxonomy of Private Web Publishing Solutions

We have introduced a new taxonomy for these services, which is depicted on Fig. 1. The leading idea during the preparation of the taxonomy was to model how the user relates to the application before starting to use it. Therefore, the first set of attributes which we chose to branch the universe of access control applications for published data were gradual deployment, realisation and ease of installation. If gradual deployment is not possible, the application is realised as external software, or its installation is complicated, we put the solution into the category 'impractical'; otherwise it is labelled as 'practical'.

The fork of the category 'practical' has been chosen to be based on the independence of the application. If the program is not at least operating system independent, or it is service specific, it is classified as 'dependent', else it is put into the category 'independent'. The reason for this is that the user is likely to prefer solutions that can be used in several environments, e.g. if she intends to run an application both on her corporate computer with Windows and her home computer running Linux.

The split on category 'independent' has been based on discoverability and key distribution, because these factors have major influence on the security properties of the software. (We have not included the type of encryption, since it does not tell much about the security properties of the application.) If the presence of an application is discoverable, i.e. its discoverability attribute is 'crypto', or key management is not automatic, the solution is classified as 'recommended'. In other cases, it is assigned the label of 'smoothly usable'.

Lastly, the fourth split considers compromises and ease of usage. These factors have a major impact on user experience, so they are likely to influence the user's relation to the software in the long run. Based on this idea, we have split the category 'recommended' into 'average' and 'good'; if the application seriously hinders the use of the host application (i.e. the Web 2.0 service it is applied on), or it is cumbersome to use, we use the former category, else we put it in the latter. The category 'smoothly usable' is split into 'powerful' and 'ideal' based on a similar reasoning.

The column 'Taxonomy type' in Table I. summarises the results of fitting the taxonomy onto the applications discussed in Section II A. It can be seen that all current solutions that are discussed in this paper are 'dependent', since they are service specific. Our own solution, besides having other merits, is service independent, and is easy to use, as can be seen in the discussion in Section III.

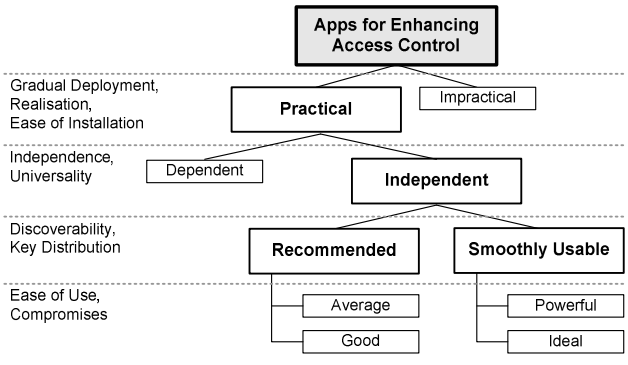


Figure 1. The taxonomy of access control solutions.

TABLE I. CLASSIFICATION OF CURRENT SOLUTIONS

Program	Gradual deployment	Inde-pendence	Universality	Compro-mises	Ease of usage	Ease of installation	Encryption	Discover-ability	Key distribution	Realisation	Taxonomy type
SeGoDoc [6]	N/A	OS	SS	50-99%	Min	NoConf	Sym	Crypto	N/A	Ext	Depend-ent
Persona for Facebook [7]	Possible	OS	SS	50-99%	Min	NoConf	Both	Crypto+ Fake	Auto	Ext	Depend-ent
FaceVPSN [8]	Possible	OS	SS	50-99%	Normal	NoConf	N/A	Fake	N/A	Ext	Depend-ent
StegoWeb	Possible	OS, B, S	GI	50-99%	Normal	NoConf	Sym	Stego+ Crypto	M	B	Good
Ideal	Possible	OS, B, S	UI	100%	Min	NoConf	N/A	Stego+ Crypto	Auto	Ext or B	Ideal

Therefore, it falls into the category 'good'. It must be noted that this result could be improved by implementing the ideas discussed in Section IV.

III. STEGOWEB: A SIMPLE BOOKMARKLET

Our solution called StegoWeb is implemented as a bookmarklet, i.e. as a simple program that can be executed by clicking on a bookmark in the browser. In this section, we describe the model on which it is based, and then provide a description of the actual implementation. The simple usability and the absence of special software requirements were fundamental aspects during the design phase.

A. Model of StegoWeb

Our model defines four separate entities:

- Browser: Operations can be controlled by the user with installed bookmarklets.
- Web service: It stores the fake data which serves as a pointer to the location of the real data on a URL shortener service (see below).
- Application storage: It stores JavaScript libraries realising the core functionality of StegoWeb.
- URL shortener service: It stores the real data in an obfuscated form. Arbitrary URL shortener service will do, provided that it supports the aliasing feature, preferably complemented by the ability to delete already registered aliases, e.g. to revoke keys.

We define the following primitives for describing the operation of the algorithms:

- `e(x, k)`: Encrypts `x` with key `k`.
- `d(x, k)`: Decrypts `x` with key `k`.
- `h(x)`: Returns the one-way hash (digest) of `x`.
- `cat(`x_1`, `x_2`, ...)`: Concatenates its arguments.
- `fetch(x)`: Returns the data field of the URL registered under the alias `x` at the URL shortener service.

The inputs of the hiding and revealing algorithms are as follows:

- `KEY`: A key for a symmetric-key encryption algorithm.
- `FAKE_URL`: The address of the website which contains the fake data.
- `REAL_DATA`: An atom of data to be hidden, corresponding to some content on `FAKE_URL`.
- `XPATHS`: The XPath expressions corresponding to the elements in `FAKE_URL` for which `REAL_DATA` is to be hidden. (In our implementation, the user has to provide these by highlighting text on the webpage.)

When performing hiding, three parameters are considered for each piece of fake data: its XPath, the corresponding original data, and the key. The operation is executed in two steps. First, the XPath expressions are hidden, and then the real data. These algorithms can be described as follows:

- XPath hiding:
```
ALIAS := h(cat(KEY, FAKE_URL))
DATA := e(XPATHS, KEY)
```
- Data hiding:
```
ALIAS := h(cat(KEY, FAKE_URL, XPATH))
DATA := e(REAL_DATA, KEY)
```

After each sub-operation, a URL *http://example.com/?data='DATA'* is registered under the alias `ALIAS` at the URL shortener service. Technically, the domain part of the address is arbitrary, but it is wise to choose a popular website to avoid attracting attention.

Real data can be revealed in two steps, too, as follows:

- Revealing XPaths:
```
ALIAS := h(cat(KEY, FAKE_URL))
XPATHS := d(fetch(ALIAS), KEY)
```
- For each entry in `XPATHS`, revealing real data:
```
ALIAS := h(cat(KEY, FAKE_URL, XPATH))
DATA := d(fetch(ALIAS), KEY)
```

The entire process of revealing the real data is depicted on Fig. 2.

B. Description of the Implementation

Our implementation is realised as a set of bookmarklets that download a short JavaScript code which realises the aforementioned operations. We have used MD5 as a hash algorithm, AES-CBC as a cipher, and *http://is.gd* as a URL shortener service.

To hide data, the user has to navigate to the website that contains the fake data, click on the selection bookmarklet, and select some text on the page (Fig. 3 (a)). The user is then prompted to enter the corresponding real data. These steps can be repeated as many times (i.e. with as many pieces of text) as desired. When finished, the user has to click on the hiding bookmarklet, which asks the user to type the key to be used for the hiding (Fig. 3 (b)).

To reveal data, the user has to go to the website containing the fake data, and click on the revealing bookmarklet. A dialog box appears where the user has to enter the key. If the right key was provided, each piece of fake data is substituted with the corresponding real data, completely automatically (Fig. 3 (c)).

C. *Analysis*

Our solution slightly deviates from the classical idea of steganography, where information is hidden directly into a cover media, the result of said operation being the stego media. However, our opinion is that classifying this technique as steganographic is appropriate, since the result of its application is that the very fact of the existence of hidden information is hidden from all unauthorised parties. In this – somewhat broader – sense, the cover and stego media can be defined as the combination of the fake website and the set of URLs registered at the shortener service. Indeed, it is impossible to tell if an alias contains hidden information without the key and the fake webpage, provided that the domain part does not give it away. In other words, StegoWeb makes the fake URL hide in the crowd of real URLs. Of course, URLs registered by StegoWeb can only accidentally lead to valid websites, but the case of a user error and that of the use of StegoWeb is not easily distinguishable by the service provider. This is a steganographic quality, even if symmetric-key encryption is used as a core idea of the algorithm. (E.g. TrueCrypt Hidden Volumes [17] are based on hiding cryptograms, too.)

The major advantage of the model is that it can be implemented with free and public web services (just like our proof-of-concept implementation), and it does not require any software to be installed. It is easy to use, and the real data is accessible only on the client side, so the solution is not dependent on the trustworthyness of third party services. Indeed, the URL shortener service and the application server can be easily replaced by other providers, as long as the former complies with the requirements discussed in Section III. A. Furthermore, it is independent both from the browser and the operating system, and therefore it can be used on any platform that can run JavaScript code, which is customary in all modern browsers.

A potential disadvantage is that bookmarklets do not run automatically once the webpage containing the fake data is

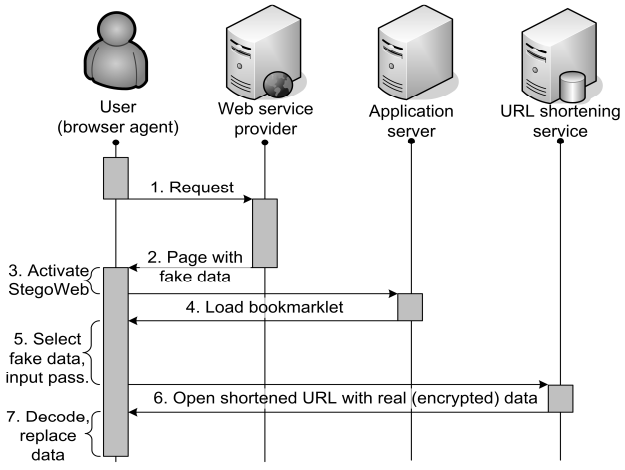
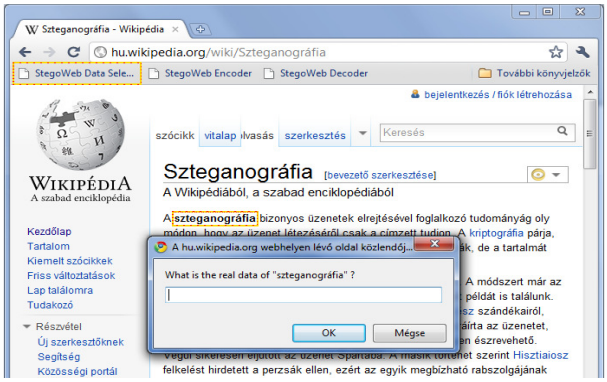
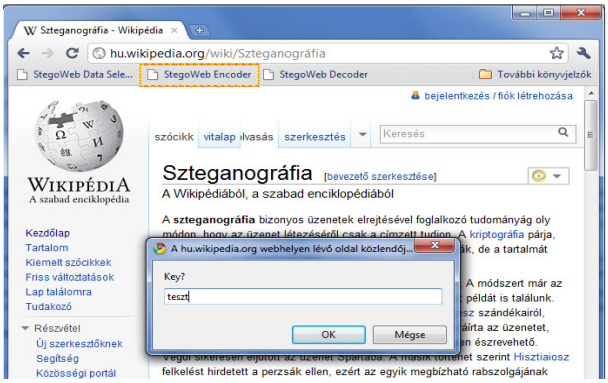


Figure 2. Revealing real data.



(a)



(b)

(c)

Figure 3. Using StegoWeb [18].

loaded, and therefore user interaction is required to reveal the real profile. Furthermore, the password cannot be stored in the browser, so it must be typed again for each execution. (This drawback will be easily eliminated when the penetration of HTML5 local databases will make efficient browser-side storage possible.) Updating the real data is also a problem given that the key has to be renewed every time. Moreover, revocation is possible only if the URL shortener supports deleting aliases, so it is wise to choose such a service that supports this feature.

It must be noted that there is a risk pertaining to the third parties (i.e. the web service, the application storage, and the URL shortener) that our solution relies on. First of all, any of these may stop functioning, e.g. if a server falls victim to a denial-of-service attack. As such, this vulnerability constitutes an availability problem. Furthermore, the scripts hosted on the application storage might be 'poisoned', as is the case when a cracker replaces the StegoWeb libraries with her malicious code. This is a classical system security risk. Fortunately, both vulnerabilities can be eliminated by using multiple application storage and URL shortener services and comparing the information obtained from all sources. Digital signatures may also be considered for additional protection.

To test the implementation, we have verified our solution with several websites. Test runs of the software can be accessed through the webpage *http://stegoweb.pet-portal.eu*.

IV. Key and Identity Management

In this section, we discuss the possible ways of managing keys and identities in our model. If HTML5 databases were widespread enough, these ideas would have made it into the final implementation. Until then, however, it does not make much sense to try to implement automatic key management into our bookmarklet, because an efficient, yet platform-independent means of client-side storage is absent.

A. Key Management

Symmetric-key algorithms use only one key for encryption and decryption (or the encryption key can easily be transformed into the decryption key and vice versa). In our model, data is stored only on the sites of URL shortener services, and, assuming a symmetric-key algorithm as a basis of the implementation, the alias depends on this single key and the URL of the page containing the fake data. Consequently, there is no obstacle to hiding information with different keys for a given profile page. This key can be unique either for each user or for a group, depending on the recipients themselves. The revocation of a key is also simple by deleting the URL alias (if such a feature is implemented on the URL shortener).

Asymmetric-key algorithms use two different keys: a public for encryption and a private for decryption. In StegoWeb, the algorithm for hiding using asymmetric encryption could be similar to the symmetric case. The main difference is that a symmetrical message key is created for each occasion of hiding, and this key is encrypted by the addressees' public keys, who get the cryptograms in private messages. (N.b., if asymmetrical keys were the bases of the encryption phase during hiding, the 'hidden' URLs would be easy to reproduce by everyone, and therefore the goal of steganography would be thwarted, hence the idea of message keys.) The revocation of a key can be performed with the deletion of the alias, in this case, too.

The main problem of asymmetric-key cryptography is the distribution of keys. This obstacle can be easily overcome for certain services; for instance, users of a social network may share this key on their profile page. A script can then collect the keys from their friends' profiles, without explicit communication with them. In other cases, a PGP-like web of trust mechanism [1] can be used for distributing keys, but this involves communication between the participating users.

B. Privacy-Enhancing Identity Management

Here we propose a Privacy-Enhancing Identity Management (PIDM) model for StegoWeb. Our model is based on a social networking service whence public keys of users can be obtained, and where the objective is to conceal profile attributes. A general PIDM scheme offers a hierarchy for managing profiles where attributes of the profiles are inherited from their ancestors in the tree structure or set by the user for the specified profile [3].

This concept can be customised for StegoWeb based on asymmetric and symmetric-key cryptography. If all users publish their public keys on their profile page, sending information (e.g. real profile data) involves looking up the public key, using the hiding algorithm as described in Section III. A, and then notifying the addressee out-of-band, e.g. through the private messaging feature of the social networking website. This works excellently for a single user, but it would require too many aliases at the URL shortener for many addressees. As such, we suggest a hierarchy of groups, each of which has its assigned symmetric key. The users themselves are at the bottom of the hierarchy, and each of them gets the secret keys of the groups that are located between the root of the tree and them when they are added. Then, if one wants to reveal her profile data to a group of users, she does the encryption part of the hiding algorithm with all the intermediate keys, and then registers the URL at the shortening service. Each addressee can then try the revealing algorithm with all key combinations she is aware of. (N.b., all of them got at least one combination of secret keys when they were added to the hierarchy, but they could have got multiple if they were included in multiple groups.)

It can be seen that the number of users in a group defines a trade-off between the number of URLs to be created by the sender and the difficulty of revoking a group key; if one defines groups with high granularity, the sender has to use many different key combinations to publish her real profile information, but 'unfriending' someone is not difficult due to the low number of users in the same group, and conversely, if a group has many users, the initial effort required from the sender is low, but rekeying the group is cumbersome if the fluctuation is high.

It is also interesting to consider the model from the perspective of plausible deniability. One can create multiple

hierarchies for the same set of social network friends, and send different profile information on both. Then, if an addressee is forced to surrender her keys, she can hand over those that lead to fake information, and deny the existence of another set of keys, provided that the fake profile information is plausible. This way, the sender's profile data can be effectively protected even from powerful third parties.

V. Conclusion and Future Work

In this paper, we have discussed the threats that profiling poses to users, and categorised the existing solutions that aim to address this problem. The basis of comparing the applications we could find in academic literature was a categorisation scheme based on important attributes that apply to virtually all such applications, and a taxonomy based on these attributes. Then we proposed an own model, and discussed our own implementation thereof. We showed that, through the realisation as a bookmarklet, our solution is not only secure thanks to the underlying steganographic principles, but it is also very easy to use, versatile and as platform independent as possible, the only requirement being a browser that can interpret JavaScript code.

As far as possible improvements are concerned, we believe that the most crucial deficiency is the lack of key and identity management. We have described some alternatives for this in Section IV. When HTML5-based local storage becomes a standard in all modern browsers, this feature can be easily implemented in a completely platform independent way, which is, we argue, paramount for such solutions. Additionally, these features can be enhanced with GUIs created with JavaScript.

Another way of implementing key and identity management would be to realise our solution as a browser extension, so that the application could use the local storage space of the browser itself. This could possibly lead to being bound to a single platform; however, implementing the application as an extension can be advantageous, because the revealing algorithm could be triggered automatically. Such a feature is suitable especially for use with social networking profiles, as these are webpages that have a more or less fixed structure, in contrast to blogs, photo sharing websites, etc. It must be noted that the extension to be implemented is very simple, so it could be easily realised for all modern browsers.

Finally, it would be interesting to verify the implementation in an experiment involving several users. This way, both user experience (e.g. ease of use) and other fundamental properties of the algorithm (e.g. its steganographic security and capacity properties) could be assessed. The results would provide important feedback about what we should improve in the implementation, and a more in-depth comparison to other similar implementations would be possible, too.

Acknowledgement

We would like to thank the High Speed Networks Laboratory for financially supporting our work.

References

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

[2] B. Krishnamurthy and C. Wills, "Privacy diffusion on the web: a longitudinal perspective," Proc. of the 18th international conference on World wide web, April 2009, pp. 541–550., doi: 10.1145/1526709.1526782.

[3] G. Gy. Gulyás, R. Schulcz, and S. Imre, "Modeling Role-Based Privacy in Social Networking Services," Proc. of the 2009 Third International Conference on Emerging Security Information, Systems and Technologies, SECURWARE, June 2009, pp. 173–178, doi: 10.1109/SECURWARE.2009.34.

[4] T. Paulik, Á. M. Földes, and G. Gy. Gulyás, "BlogCrypt: Private Content Publishing on the Web," Proc. of the 2010 Fourth International Conference on Emerging Security Information, Systems and Technologies, SECURWARE, July 2010, pp. 123–128, doi: 10.1109/SECURWARE.2010.28.

[5] Y. Zhu, Z. Hu, H. Wang, H. Hu, and G-J. Anh, "A Collaborative Framework for Privacy Protection in Online Social Networks," Cryptology ePrint Archive: Report 2010/491, 2010.

[6] D'Angelo, G., Vitali, F., and Zacchiroli, S, "Content Cloaking: Preserving Privacy with Google Docs and other Web Applications," Proc. of the 25th Annual ACM Symposium on Applied Computing, March 2010, pp. 826–830, doi:10.1145/1774088.1774259.

[7] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: an online social network with user-defined privacy," Proc. of the ACM SIGCOMM 2009 conference on Data communication, August 2009, pp. 135–146, doi: 10.1145/1592568.1592585.

[8] M. Conti, A. Hasani, and B. Crispo, "Virtual private social networks," Proc. of the first ACM conference on Data and application security and privacy, February 2011, pp. 39–50, doi: 10.1145/1943513.1943521.

[9] NOYB: Posting Secret Messages on the Web, http://adresearch.mpi-sws.org/noyb.html, retrieved on March 27th, 2011.

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

[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] 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

[14] 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.

[15] evercookie - virtually irrevocable persistent cookies, http://samy.pl/evercookie/, retrieved on June 1st, 2011.

[16] Panopticlick, https://panopticlick.eff.org/, retrieved on June 1st, 2011.

[17] TrueCrypt, http://www.truecrypt.org/, retrieved on June 1st, 2011.

[18] StegoWeb, http://stegoweb.pet-portal.eu/index_en.html#usage, retrieved on June 1st, 2011.