2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks

Social Puzzles: Context-Based Access Control in Online Social Networks

Murtuza Jadliwala, Anindya Maiti and Vinod Namboodiri
Wichita State University, Wichita, Kansas, 67260
Email: {murtuza.jadliwala, axmaiti, vinod.namboodiri}@wichita.edu

***Abstract*—The increasing popularity of online social networks (OSNs) is spawning new security and privacy concerns. Currently, a majority of OSNs offer very naive access control mechanisms that are primarily based on static access control lists (ACL) or policies. But as the number of social connections grow, static ACL based approaches become ineffective and unappealing to OSN users. There is an increased need in social-networking and data-sharing applications to control access to data based on the associated context (e.g., event, location, and users involved), rather than solely on data ownership and social connections. Surveillance is another critical concern for OSN users, as the service provider may further scrutinize data posted or shared by users for personal gains (e.g., targeted advertisements), for use by corporate partners or to comply with legal orders. In this paper, we introduce a novel paradigm of context-based access control in OSNs, where users are able to access the shared data only if they have knowledge of the context associated with it. We propose two constructions for context-based access control in OSNs: the first is based on a novel application of Shamir's secret sharing scheme, whereas the second makes use of an attribute-based encryption scheme. For both constructions, we analyze their security properties, implement proof-of-concept applications for Facebook and empirically evaluate their functionality and performance. Our empirical measurements show that the proposed constructions execute efficiently on standard computing hardware, as well as, on portable mobile devices.**

***Keywords*-Online Social Networks, Access Control, Privacy, Surveillance Resistance.**

I. Introduction

An online social networking (OSN) service is a popular tool for online users to connect with other users who are either real-life acquaintances or have similar interests and background. The Wall Street Journal reported that *Facebook's*[1] user base had increased to one billion users at the end of 2012 [1]. OSN services allow its users to maintain a profile, update personal information and share pictures, posts, activities, events, and interests with other users in their social network. The *privacy* of personal and shared information, with respect to the service provider and other users, is of paramount importance to OSN users [2].

In order to provide privacy with respect to other users, OSNs enforce access control policies on the data being shared, wherein, only a specific set of receivers dictated by the policy can get access to a user's personal and shared information. Existing OSN access control mechanisms are based on either *static policies* (for example, by default all users in the friend list are allowed to view all posted images) or *fine-grained ACLs* where specific groups within a user's social network are allowed access to specific categories of the user's information [3]. These access control mechanisms are mostly *user-centric*, rather than information or *data-centric*. In a recent study [4] of over 250 users, it was found that while strangers or non-friends are the most concerning audience when it comes to sharing data on Facebook, most users take appropriate steps to mitigate those concerns. However, 16.5% of the participants had at least one post that they were uncomfortable sharing with a specific friend - someone who likely already had the ability to view it - and 37% raised more general concerns with sharing their content with friends. The study concludes that, although Facebook privacy controls are effective against threat from non-friends, they are unsuitable for the insider threat (i.e., from friends) who dynamically become inappropriate audiences based on the context of a post. Thus, there appears to be a need in OSNs to dynamically share data based on the *knowledge (or context)* related to the data being shared.

We envision a new paradigm of dynamic access control in OSNs, called *social puzzles*, which performs access control based on the knowledge of the shared data and the context related to it. Nearly all content shared on OSNs is related to past, present or future events, where each event is associated with a unique context involving location, time, activities, participants and preferences. Individuals involved in an event are presumed to have gained knowledge of the related context and some of this context may remain the same for future similar events. This makes sharing data related to events, of which the associated context is presumed to be known by the intended audience, a suitable proposition. An example of this includes sharing messages or pictures of a past social gathering involving the target audience (who are also friends on an OSN). The idea of context-based data sharing is not only restricted to OSNs, but can also be applied to other data sharing services such as microblogging services (e.g., Tumblr), photo sharing services (e.g., Picasa and Instagram) and file storage and sharing services (e.g., Dropbox and OneDrive). Other customized applications can also be envisioned, e.g., data management in a corporate network, where only employees knowing certain work-related context can get access to certain confidential documents.

Our goal in this work is to design access control mech-

[1]Facebook, https://www.facebook.com

978-1-4799-2233-8/14 $31.00 © 2014 IEEE
DOI 10.1109/DSN.2014.38

anisms for OSNs using dynamic context-based policies, which not only seamlessly integrate with existing OSNs, but also provide resistance against surveillance by service providers. These new access control mechanisms will complement existing static policies on OSNs, thus providing users with additional flexibility while sharing data, and will improve privacy of the shared data without compromising the utility of the OSN service. One advantage of our proposed mechanisms is that it will simplify the access control process in undirected OSNs such as Facebook and *Google+*[2], especially when access control is required to be based on the knowledge of the context surrounding the data. In order to provide such an access control, service provider and users will no longer be required to maintain complex and constantly mutating and expanding access control lists (ACL). Moreover, OSNs with directed social connections and the ones that provide only very minimalistic access control mechanisms (e.g., *Twitter*[3]) will benefit even more because the context-based access mechanism will add a layer of privacy protection.

As access control in OSNs is currently either performed by the service provider itself, or in some cases by a trusted third-party [5], [6], [7], they typically have access to the data being shared or the access control policy used to share the data. This is not a desirable situation for users who do not trust the service provider (or third-parties) and want security against release of data or access control policies to these parties. Our proposed access control mechanism only trusts the service provider to execute the access control protocol honestly; the service provider is able to perform access control operations without the knowledge of the data being shared or the context based on which access control is done. Such a *surveillance-resistance* property can prevent service providers from mining user data for gaining corporate advantage (e.g., targeted advertisements) or sharing it with other entities (e.g., government monitoring programs and corporate partners) without user consent.

Another advantage of the proposed context-based access control mechanisms is improved content-relevance. With an increasingly large number of online social contacts, OSN users typically find themselves bombarded with or buried under a large amount of irrelevant or sparsely relevant information from their contacts. Good access control inherently leads to better content-relevance for OSN users. Context-based social routing [8] is another effort in this direction, where each user specifies their interests through a set of keywords and the data routing algorithm routes shared objects with relevant context attributes to the users. We argue that our context-based access control mechanism will inevitably enforce relevant content being read, because users cannot access contents with unfamiliar contexts.

[2]Google+, https://plus.google.com

[3]Twitter, https://www.twitter.com

Finally, we anticipate that any new access control mechanism should be easy to use, else users may continue to settle for inferior privacy settings. A trivial context-aware access control scheme can be constructed as follows: sharer generates a symmetric encryption key (and then encrypts data) by using *all* the context associated with the data, while the receiver regenerates the key (to decrypt the data) by proving knowledge of the *entire* context. However, such a trivial scheme is not useful because most of the times receivers will not be aware of the entire context related to the shared data. The proposed mechanisms are much more flexible and allow the sharer to specify a *threshold* on the *number* or amount of context required to be known by the receivers before they can access the data. Thus, receivers can access data by proving only partial knowledge of the related context. We realize that when it comes to usability, systems that require complex setup and regular maintenance [9], [10] are not convenient and/or popular. Thus, our goal is to design and implement mechanisms that incur low performance overhead, require little or no maintenance and can be easily integrated with popular services such as Facebook.

The key contributions of this paper are as follows:

- We propose two novel constructions for context-based access control to enable private data sharing among OSN users.
- We demonstrate the feasibility of our constructions by developing a publicly-available[4] proof-of-concept implementation for Facebook. A careful security analysis under various adversarial scenarios is also performed.
- We empirically evaluate the functionality and performance of our implementation for a variety of operational parameters.

II. Related Work

Each OSN service addresses the problem of data privacy and access control differently. Facebook, for example, provides customized ACLs where access to a particular data object is restricted to only those social contacts that are present in a user's ACL. One shortcoming of ACLs is that they are not very scalable. Increasingly large, and highly dynamic, list of friends or social contacts can lead to a burdensome maintenance of such access lists. In Twitter, on the contrary, there are no privacy constraints and all tweets are public (by default). Few other researchers have studied the possibility of role-based [11] and attribute-based [12] access control in OSNs. But both these schemes require additional infrastructure and support from the OSN provider, thus making them less likely to be adopted in practice. Contrary to these, our proposed access control mechanisms can be hosted either by the OSN provider or by some other third-party provider. In our schemes, much of the access

[4]http://socialpuzzle.cs.wichita.edu/

control functionality is performed locally on the client on an on-demand basis, which is more efficient.

A majority of the OSN providers have a "default open" policy, wherein a lot of sensitive and personal information about subscribers is available or easily accessible by all other subscribed users. Security and privacy of user data in OSNs has received significant attention in the literature, but it still remains an open problem [13], [6], [14], [15], [16]. Existing research efforts have primarily focused on: (a) decentralized OSNs, (b) dedicated infrastructure at the end-user, (c) access control by trusted third-party, and (d) secure data sharing using public-key cryptography.

Yeung et al. [13] propose a decentralized approach to online social networking where each user possesses a trusted server which stores user-data and enforces pre-defined access control policies. Anyone trying to access a user's personal data is redirected to the trusted server which first makes an access control decision. Similarly, in order to view protected shared objects in *Diaspora*[5], friends would need to access the user's personal web server or start using the Diaspora service. Dürr et al. [10] propose another decentralized OSN, called *Vegas*, which allows secure information sharing with *nearby* users. In another related effort, Jagtap et al. [9] propose a data-on-demand list-based social networking methodology which abstracts sensor data of mobile devices by using a "*Privacy Control Module*" located on the user's device. Decentralized mechanisms, however, are not compatible with existing OSN services and they require individual users to possess dedicated infrastructure for data storage and access control. Alternatively, a semi-decentralized architecture proposed by Carminati et al. [17] reduces client-side workload and infrastructure requirement, but it needs to be continuously available. Compared to these, our context-based access control mechanisms are designed to work with existing OSN services, without requiring additional hardware infrastructure and with limited resources (ideal for resource-constrained mobile devices). To eliminate the dependence on client-side infrastructure for access control, a few of the above schemes propose the use of a trusted third-party. Although the third-party can be trusted to perform access control correctly, it still needs access to user's policies and data, which could open door to surveillance. In our proposal, we also require a third-party (or OSN) to host the context-based access control service, but our schemes are resistant to surveillance by these providers or third-parties.

In order to guarantee confidentiality against service providers, data has to be encrypted at the client-side (by the sharer) such that only the intended receiver(s) are able to decrypt it. Beato et al. [14] achieve this by using OpenPGP. The authors propose a scheme that is hybrid between the trusted server and the decentralized approaches. Although their implementation integrates well with existing OSNs, the usability of their scheme suffers due to the required public-key management operations. FaceCloak [6] secures Facebook profile data and messages by using symmetric-key encryption and storing them on a separate server, while posting fake information on the actual Facebook profiles. Contrary to this, the scheme by Beato et al. [16] achieves privacy by using asymmetric encryption and by anonymizing user information from the shared data objects. Earlier, Beato et al. [15] proposed a service provider-independent scheme, called Scramble!, to assure confidentiality and integrity of OSN user data. The authors implement a browser extension that allows users to enforce access control over their data, as well as, protect it against surveillance and modification from service providers. Both FaceCloak and Scramble! are not very easy to use because every friend has to actively exchange and maintain a set of valid encryption/decryption keys. In contrast, our mechanisms do not require periodic and expensive key exchanges. Moreover, we address the problem of access control based on the knowledge of the shared data, and not based on the users in the social network. Most importantly, due to its JavaScript-based implementation, only a standard web browser (without any additional installation/configuration) is required for using our scheme.

[5]Diaspora, http://www.joindiaspora.com

III. Basics and Background

Before going into the details, let us briefly outline a few well-known cryptographic constructions that we use in our proposals. Our first construction employs Shamir's secret sharing scheme (section III-B), whereas our second construction uses an attributed-based encryption scheme such as CP-ABE (section III-C). The mathematical notions of bilinear maps and pairings (section III-A) are useful for understanding CP-ABE.

A. Bilinear Maps and Bilinear Pairing

Let $\mathbb{G}_0$, $\mathbb{G}_1$ and $\mathbb{G}_2$ be multiplicative cyclic groups of prime order p. Let g_0 and g_1 be generators of $\mathbb{G}_0$ and $\mathbb{G}_1$, respectively. Let e be a *bilinear map* from $\mathbb{G}_0 \times \mathbb{G}_1$ to $\mathbb{G}_2$, i.e., e is a function $e : \mathbb{G}_0 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, such that for all $u \in \mathbb{G}_0$, $v \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$ (*bilinearity* property) and $e(g_0, g_1) \neq 1$ (*non-degeneracy* property). If $\mathbb{G}_0 = \mathbb{G}_1$, then the pairing e is symmetric. This is because, $e(g_0^a, g_0^b) = e(g_0, g_0)^{a,b} = e(g_0^b, g_0^a)$.

B. Shamir's Secret Sharing Scheme

In Shamir's (k, n) threshold secret sharing scheme [18], a secret is shared among a set of, say n, participants by dividing it into parts (or shares) such that each participant possesses a unique share. The secret can then be reconstructed from a threshold, say k, number of shares obtained from the participants. Let's assume that we want to share a secret M, where M is an element in the finite field $\mathbb{F}$ of size p (p is a prime s.t. $0 < k \leq n < p$). We create a random polynomial $P \in \mathbb{F}(x)$ of degree k by choosing $k-1$ random

coefficients in $\mathbb{F}$ and $P(0) = M$. Each of the $i = 1 \dots n$ participant receives the share $(i, P(i))$. Now, given any k of these shares $P(s_1), P(s_2), \dots, P(s_k)$, where $s_j \neq s_{j'}$ and $s_j, s_{j'} \in \{1 \dots n\}$, the secret $P(0) = M$ (constant term of P) can be recovered using Lagrange interpolation as:

$$P(0) = \sum_{j=1}^{k} \gamma_j P(s_j), \text{ where } \gamma_j = \prod_{j' \neq j} \frac{s_{j'}}{s_{j'} - s_j}$$

C. Ciphertext-Policy Attribute-Based Encryption (CP-ABE)

In *ciphertext-policy attribute-based encryption (CP-ABE)* [19], a party encrypting a message can specify a policy (based on attributes describing user credentials) for who can decrypt. Specifically, the private key (used to decrypt) is associated with an arbitrary number of attributes. When a party encrypts a message, he specifies an access structure over these attributes. Any user can decrypt this message only if his attributes pass through the ciphertext's access structure. CP-ABE consists of the following main procedures.

- **Setup**: The *Key Authority (KA)* takes no other input, except a security parameter, and outputs a public key $PK = \mathbb{G}_0, g, h = g^{\beta}, f = g^{\frac{1}{\beta}}, e(g,g)^{\alpha}$ and a master secret $MK = (b, g^{\alpha})$, where $\mathbb{G}_0$ is a bilinear group of prime order p and $\alpha, \beta \in \mathbb{Z}_p$.
- **Encrypt**(PK, M, τ): This algorithm encrypts a message M under a policy τ which is represented as a tree access structure by using the public key PK. Let s (a random number in $\mathbb{Z}_p$) be the secret at the root of the policy tree, q_x be the polynomial of degree $d_x = k_x - 1$ at the node x where k_x is the threshold value at the node x, Y be the set of leaf nodes in τ and **att**(y) returns the attribute of the leaf node y. The ciphertext CT is:

$$CT = (\tau, \tilde{C} = Me(g,g)^{\alpha s}, C = h^s, \\ \forall y \in Y : C_y = g^{q_y(0)}, C'_y = H(\textbf{att}(y))^{q_y(0)})$$

  Here, H is a hash function that maps to a random element in $\mathbb{G}_0$, i.e., $H : \{0,1\}^* \rightarrow \mathbb{G}_0$
- **KeyGen**(MK, $\mathbb{S}$): It takes as input a set of attributes $\mathbb{S}$, the master secret MK and outputs a key that identifies with that set. It chooses randoms $r \in \mathbb{Z}_p$, and $r_j \in \mathbb{Z}_p$ for each attribute $j \in \mathbb{S}$ and computes the key as:

$$SK = (D = g^{(\alpha + r)/\beta}, \\ \forall j \in \mathbb{S} : D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{r_j})$$

- **Decrypt**(CT, SK, x): It implements a recursive algorithm **DecryptNode**(CT, SK, x). For each leaf node x in τ, **DecryptNode** pairs D_i and D'_i (from SK) with C_x and C'_x, resp., to obtain $e(g,g)^{rq_x(0)}$ if $i \in \mathbb{S}$, where $i =$ **att(x)**. If $i \notin \mathbb{S}$, then **DecryptNode** returns $\perp$. For each non-leaf node x in τ, it recursively calls **Decrypt**(CT, SK, z_j) on all children z_j of x. It then calculates $e(g,g)^{rq_x(0)}$ for the non-root node x by using Lagrange interpolation on at least k_x such $e(g,g)^{rq_{z_j}(0)}$ obtained from its children $\{z_j\}$. If k_x such $e(g,g)^{rq_{z_j}(0)}$ are not available then **DecryptNode** returns $\perp$ for the non-root node x. **Decrypt** begins by calling **DecryptNode**(CT, SK, R) on the root node R and computes $A = e(g,g)^{rq_R(0)} = e(g,g)^{rs}$. It then retrieves M by computing $\tilde{C}/(e(C,D)/A)$.

IV. Model

In this section, we outline details of the system and adversary model considered in this work.

A. The System

We consider an *OSN* provider, denoted by SP, where each subscribing user maintains a list of contacts (or friends) and uses the OSN platform to store and share digital content (e.g., status updates, photos, locations, etc.) with his/her *social network*. We consider a *symmetric* social networking service, i.e, if a user a has another user b in her friend list, then user b has user a as her friend as well. A popular example of a symmetric OSN is Facebook. OSN services usually maintain a *profile* and a *list of contacts* (which includes relationship type) for each registered user. A profile typically contains personal information which uniquely identifies the user. Users can typically add/update their profile information and access profiles of their contacts at various levels of granularity, often dictated by the contact's privacy setting.

We consider a user S, referred to as the *sharer*, who has a registered account with the service provider SP. The sharer S wants to share some data object O, e.g., a picture or video file, with her contacts (or social network) S_T, provided they have some knowledge of the *context* related to the sharer and/or the object O. S is unwilling to share the object with those contacts who may not know this context. There may also be contacts who may not be interested in receiving certain data objects from the sharer without knowing the related context. Such situations are very common in social networking or other data sharing applications. For instance, a user may want to share pictures of a particular private event with only those contacts who were either at the event or were invited but missed the event. Some other contacts, such as professional contacts, may not be interested in viewing the sharers' personal event pictures or it may be inappropriate for the sharer to share those with them.

The context C_O related to an object O can be formulated as a set of N *key-value* (or question-answer) pairs $\{\langle q_1, a_1\rangle, \langle q_2, a_2\rangle, \dots, \langle q_N, a_N\rangle\}$. Without loss of generality, let us assume that each context can be represented by *exactly* N key-value pairs. Each key q_i typically defines a domain with the corresponding a_i taking exactly a single value (from that domain). For each shared object O, the sharer S typically sets a *threshold* ζ_O on the minimum number of key-value pairs that should be known to the receiver before she can access O. Let $R_O \subseteq S_T$ be the

set of S's contacts that know the a_i values corresponding to at least ζ_O q_is related to an object O. Thus, users in R_O are said to "*know*" the context. Without loss of generality, let's assume that $\zeta_O = k$, for some $k < N$ for all objects O.

The shared object O is stored in an *encrypted form* on a *storage service* denoted as DH. Details of the encryption strategy will be clear soon. The storage service DH is logically separate from SP, but physically, it can either be co-located with the SP or hosted by a different third-party provider such as Dropbox[6]. The encrypted object stored on the DH is publicly accessible by means of a unique URI or web resource locater denoted as URL_O.

B. The Adversary

We want to protect against the unauthorized disclosure of the sharer's object O to the following entities: (i) all users (including users in the sharer's social network $S_T - R_O$) who do not know the context (have knowledge of less than ζ_O key-value context pairs), and (ii) the SP and the DH, if they do not know the context. Although our scheme is general enough and can protect against any entity that does not know the context, we focus here only on those users that belong to the sharer's social network. We rely on the social network service's access control policies to protect against users outside the sharer's social network.

Access (or non-access) for an object O granted to users in the sharer's social network, based on amount of context known, is referred to as the *access control* property, whereas, preventing disclosure of the object O to the hosting services, such as SP and DH, is referred to as the *surveillance resistance* property. Context-based access control and surveillance resistance are the two main requirements of the proposed system. We assume that users who do not know the context, i.e., all users in $S_T - R_O$, can collude with each other. However, there is no collusion between users in R_O and users in $S_T - R_O$, as otherwise the access control property can be trivially compromised.

We also assume that entities who do not know the context, including SP and DH, do not perform *active attacks*. One example of such an attack is compromising accounts of users in R_O in order to obtain context C_O related to an object O. All entities that desire access to the object O will first interact with the access control protocol who will verify the context known by the entity, and accordingly, either enable or disable access to O. Malicious users (except, SP and DH) may attempt to circumvent the operation of the access control protocol by manipulating inputs to the protocol or learn from the intermediate outputs. Finally, we assume that the SP and DH execute the access control protocol *truthfully*. Due to business and legal consequences resulting from malicious behavior, such a *semi-honest* model is a practical assumption for the service provider.

[6]Dropbox, https://www.dropbox.com

V. Construction

We propose two novel constructions for context-based access control in OSN services. These mechanisms are implemented as puzzles, referred to as *social puzzles*, where users willing to access a particular object are presented with a series of questions based on the context related to the object. Only those users that know the context (i.e., *solve* the puzzle) are able to access the object. Our first construction makes an elegant use of Shamir's secret sharing scheme [18], while the second construction uses an attribute-based encryption scheme such as CP-ABE [19].

A. Construction 1

Let $\mathbb{F}$ be a finite field of size p (where, p is a prime). Let, H be a cryptographically secure hash function and let E be a secure symmetric cryptosystem. Our first construction (Fig. 1) consists of the following subroutines.

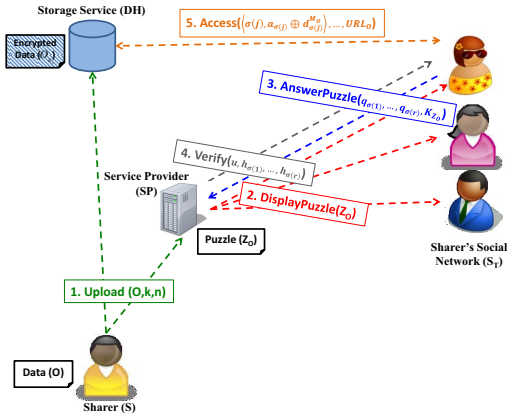


Figure 1. Construction 1

- **Upload** (O, k, n): The **Upload** subroutine is executed by the sharer S to create a social puzzle and to securely upload the object O on the storage service DH. S first determines a k and $n \leq N$ such that $0 < k \leq n < p$. Recollect that k is the minimum number of key-value pairs that should be known to a user before he/she can access the object O and N is the maximum number of context key-value pairs available for the object O. S then creates a random polynomial $P \in \mathbb{F}(x)$ of degree k by choosing $k-1$ random coefficients in $\mathbb{F}$ and a random constant term $P(0)$. The constant term of the polynomial P is used as the object-specific secret by S, i.e., $P(0) = M_O$. S then computes the object-specific secret encryption key K_O by using a cryptographically secure hash function H, i.e., $K_O = H(M_O)$. S then encrypts O using the key K_O, i.e, $O_{K_O} = E(O, K_O)$, and stores the encrypted object O_{K_O} on the storage service DH at location URL_O. S then prepares n random shares of $M_O = P(0)$ as $d_1^{M_O} = \langle s_1, P(s_1) \rangle$,

$d_2^{M_O} = \langle s_2, P(s_2)\rangle$, ..., $d_n^{M_O} = \langle s_n, P(s_n)\rangle$, where each s_i is chosen at random. S then creates a social puzzle Z_O for controlling access to O by using the context C_O of O. Specifically, the puzzle Z_O is formed using exactly $n \leq N$ question-answer pairs $\{\langle q_i, a_i\rangle\} \subseteq C_O$ and a puzzle specific key K_{Z_O}, and is shown below:

$$Z_O = \begin{Bmatrix} \langle q_1, H(a_1, K_{Z_O}), a_1 \oplus d_1^{M_O}\rangle, \\ \langle q_2, H(a_2, K_{Z_O}), a_2 \oplus d_2^{M_O}\rangle, \ldots, \\ \langle q_n, H(a_n, K_{Z_O}), a_n \oplus d_n^{M_O}\rangle, \\ n, k, K_{Z_O}, URL_O \end{Bmatrix}$$

S then uploads Z_O to the service provider SP.

- **DisplayPuzzle** (Z_O): For each puzzle Z_O, **DisplayPuzzle** is executed by the OSN provider SP for all the users u in S's social network, i.e., $\forall u \in S_T$. The subroutine first randomly picks an integer $r : k \leq r \leq n$. For the object O uploaded by S, SP selects a permutation σ of numbers from 1 to r and displays $q_{\sigma(1)}, q_{\sigma(2)}, \ldots, q_{\sigma(r)}, K_{Z_O}$ to all users $u \in S_T$.
- **AnswerPuzzle** ($q_{\sigma(1)}, q_{\sigma(2)}, \ldots, q_{\sigma(r)}, K_{Z_O}$): On receiving the puzzle questions $q_{\sigma(1)}, q_{\sigma(2)}, \ldots, q_{\sigma(r)}$, each user $u \in S_T$, if she wishes to access the object O, responds with the hash of the answers to the corresponding questions, i.e., $h_{\sigma(1)} = H(a'_{\sigma(1)}, K_{Z_O}), h_{\sigma(2)} = H(a'_{\sigma(2)}, K_{Z_O}), \ldots, h_{\sigma(r)} = H(a'_{\sigma(r)}, K_{Z_O})$. Obviously, if the user knows the correct answer $a_{\sigma(j)}$ to a question $q_{\sigma(j)}$, then $H(a'_{\sigma(j)}, K_{Z_O}) = H(a_{\sigma(j)}, K_{Z_O})$.
- **Verify** (u, $h_{\sigma(1)}, h_{\sigma(2)}, \ldots, h_{\sigma(r)}$): In response to a puzzle Z_O and displayed questions $q_{\sigma(1)}, q_{\sigma(2)}, \ldots, q_{\sigma(r)}$, SP receives $h_{\sigma(1)} = H(a'_{\sigma(1)}, K_{Z_O}), h_{\sigma(2)} = H(a'_{\sigma(2)}, K_{Z_O}), \ldots, h_{\sigma(r)} = H(a'_{\sigma(r)}, K_{Z_O})$ from a user $u \in S_T$. For each j $(0 \leq j \leq r)$, SP verifies if $h_{\sigma(j)} = H(a_{\sigma(j)}, K_{Z_O})$. If at least k such verifications are successful, then SP sends $\langle \sigma(j), a_{\sigma(j)} \oplus d_{\sigma(j)}^{M_O}\rangle$ for each correctly answered question $q_{\sigma(j)}$ to u. In addition to this, SP also sends URL_O to u. Otherwise, if less than k verifications are successful, then SP does not send anything and ends the protocol with the user u.
- **Access** ($\langle \sigma(j), a_{\sigma(j)} \oplus d_{\sigma(j)}^{M_O}\rangle, \ldots, URL_O$): On receiving URL_O, the user $u \in R_O$ downloads the encrypted object O_{K_O} from URL_O. The user u further obtains the k shares $d_{\sigma(j)}^{M_O}$ by computing $a_{\sigma(j)} \oplus (d_{\sigma(j)}^{M_O} \oplus a_{\sigma(j)})$. Once k shares are obtained, u reconstructs the object-specific secret M_O by using Lagrange basis polynomials as discussed in III-B. Once the object-specific secret M_O is reconstructed, u computes $K_O = H(M_O)$ and obtains the object $O = D(O_{K_O}, K_O)$, where D is the decryption function of the symmetric cryptosystem.

B. Construction 2

Our second construction is shown in Fig. 2. As we have already summarized CP-ABE in III-C, here we only outline how we utilize CP-ABE to construct efficient social puzzles. Before providing details, let us describe the *access tree*

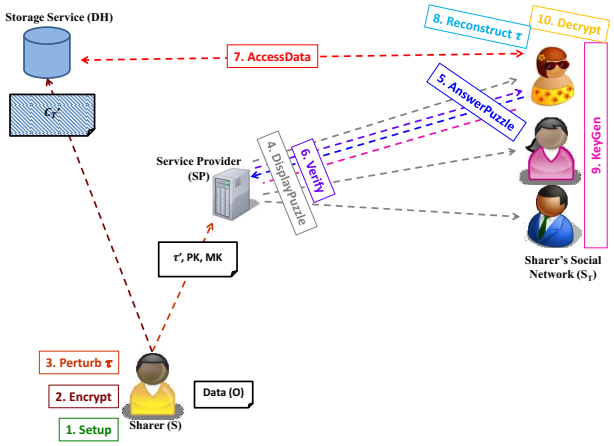


Figure 2. Construction 2

used in this construction. In CP-ABE, an access control policy is encoded as an access tree where *non-leaf* nodes are represented by *threshold gates*. Each threshold gate is described by its children and a *threshold value* (threshold value is less than or equal to the number of children). The leaf nodes describe attributes and have a threshold of one. A leaf node is *satisfied* if the attribute input by the user matches the attribute assigned to the leaf node. A non-leaf node is satisfied when at least a threshold of its children nodes are satisfied. An access tree is satisfied if and only if the root node is satisfied. Further details can be found in [19]. We now present the details of our second construction.

As before, let's assume that S wants to share an object O with all users in her social network S_T who know the context C_O about that object. Let the context C_O be defined by a total of N question-answer pairs $\{\langle q_1, a_1\rangle, \langle q_2, a_2\rangle, \ldots, \langle q_N, a_N\rangle\}$. The sharer S chooses a threshold k for the object O and creates an access tree structure τ. The access tree τ is a *monotonic tree structure* of *height* 1, as shown in Fig. 3, with a root node and N leaf nodes, where the attributes of each leaf node $\tau_{q(i)}$ are set to the values q_i and a_i. The sole purpose of τ is to enable S to encrypt object O with context attributes (q_i, a_i). S encrypts the object O using the CP-ABE **Encrypt**(PK, O, τ) routine to produce the ciphertext CT, as outlined in III-C.

In order to effectively use CP-ABE in our proposal, a minor tweak is required. The access tree τ in our proposal is first perturbed by replacing the answer attributes a_i with the corresponding hash values $H(a_i)$ to produce the *perturbed access tree* τ', as illustrated in Fig. 4. Then, S replaces the access tree τ encoded in the cipher text CT by the perturbed access tree τ' to produce the ciphertext CT'. As we will see later, this is done to prevent the DH from learning the actual access tree τ containing the answers (required to know the

context C_O). S then uploads τ', public key PK and master key MK to SP, and CT' to DH. SP shares PK and MK with all users, including the S_T.

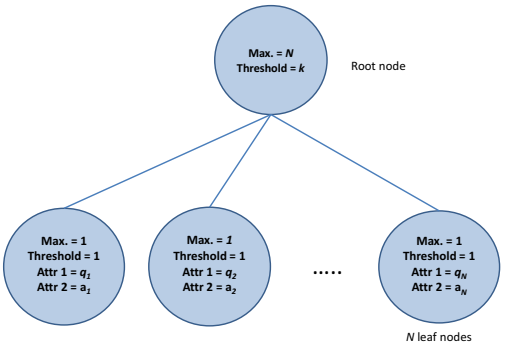


Figure 3. Access tree structure τ for Construction 2

SP then displays the questions q_i in τ' to users in S_T. On viewing these questions, users in S_T can choose to respond with a set $\mathbb{S}'$ containing the corresponding hashed answer attributes. The SP matches the hashed answers in $\mathbb{S}'$ (sent by some user in S_T) to the hashes of answers in τ'. If the number of matches satisfies the threshold ζ_O $(= k)$, the SP replies back with URL_O that points to CT' stored in DH. After downloading CT', the receiving user in S_T attempts to (partially) reconstruct τ (from τ' in CT'), by replacing at least k hashed answer attributes $H(a_i)$ in τ' with their respective real answers a_i. This *reconstructed access tree* is denoted by $\hat{\tau}$. The receiver replaces τ' in the CT' obtained from the DH with $\hat{\tau}$ to obtain $\hat{CT}$ (or reconstructed CT).

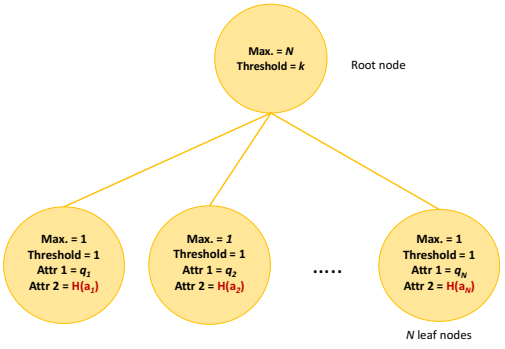


Figure 4. Pertubed access tree structure τ'

The receiver then runs the publicly known **KeyGen**(MK, $\mathbb{S}$) algorithm with the real answer attribute set $\mathbb{S}$ and master key MK to obtain the private key SK that identifies with the set $\mathbb{S}$. With $\hat{CT}$ and SK computed from the previous step, users can run **Decrypt**($\hat{CT}$, SK) subroutine to reveal the original shared object O. Below, we only describe the two new algorithms in our construction 2, namely, **Perturb** and **Reconstruct**, because the CP-ABE algorithms **Setup**, **Encrypt**, **KeyGen** and **Decrypt** are used without any changes. Moreover, **DisplayPuzzle**, **AnswerPuzzle** and **Verify** operate in a similar fashion as the first construction.

- **Perturb** (τ): It takes as input a monotonic tree τ of height 1 with N leaf nodes, and each leaf node containing one question attribute q_i and the corresponding answer attribute a_i. It replaces the a_i in each leaf node with its cryptographically secure hash value $H(a_i)$. The resulting perturbed tree τ' is sent to SP and embedded in cipher text CT' which is sent to the DH.
- **Reconstruct** (τ'): On receiving the perturbed tree τ', this algorithm (partially) reconstructs τ by correctly replacing at least k $H(a_i)$s with their corresponding a_i. This reconstructed tree $\hat{\tau}$ then replaces τ' in cipher text CT', which can then be used for decryption.

VI. Security Analysis

In this section, we analyze our constructions under *semi-honest* and *malicious* adversarial scenarios. As the two constructions are similar, except the secret key reconstruction and encryption/decryption part, we initially focus only on construction 1. Later, we outline major differences from the security perspective with construction 2.

A. Adversarial Service Provider

For a puzzle Z_O, first let us consider the semi-honest case where SP honestly executes the protocol, but wants to reveal the object O. As SP knows URL_O, she can download the encrypted object O_{K_O} from the storage service DH. If the SP knows the context C_O, i.e., knows at least $\zeta_O = k$ answers to the puzzle, she like any other user in R_O can reconstruct the encryption key K_O and reveal the object O. But if SP does not know the context, she will be unable to reconstruct the key K_O (and thus unable to reveal the object O) due to the information-theoretic security of the Shamir's secret sharing scheme. Due to the cryptographic security of the hash function H, the SP is unable to recover the individual answers a_i from the hash values $H(a_i, K_{Z_O})$ (both, the ones provided by the sharer in Z_O and the replies received from the receivers) and K_{Z_O}. Moreover, as the shares $d_i^{M_O}$ are randomly generated, and unknown to the SP, she is unable to recover the individual answers a_i from the blinded values $a_i \oplus d_i^{M_O}$. The SP does not receive any other information, either directly or indirectly due to protocol execution, from the participants which could be used to decrypt the encrypted object O_{K_O}.

Next, let us consider a few scenarios where the SP behaves maliciously. As O is not located on the SP, she cannot remove it. But, she can modify URL_O in Z_O to cause a *denial of service* (DOS). Such DOS attacks can be prevented by signing the URL_O in Z_O with the sharer's private key, which can be verified by the recipients before downloading the encrypted object O_{K_O}. The SP can also

modify the questions q_i and the puzzle-specific key K_{Z_O} resulting in a denial of service. For example, modifying K_{Z_O} will just change the hash values $H(a_i, K_{Z_O})$ provided by the receivers, and will give no advantage to the SP in recovering a_i from the corresponding hash values. Such unauthorized modifications by the SP can be overcome by including the sharer's signature for each of these components within Z_O.

B. Adversarial Storage Service

Similar to the SP, the DH cannot recover the object O from the encrypted object O_{K_O} without the secret encryption key K_O (or the object-specific secret M_O). Moreover, the security of Shamir's secret sharing scheme will prevent the DH from reconstructing the secret M_O, and thus the encryption key K_O, without the knowledge of the context C_O. However, a malicious DH can tamper, remove or modify O_{K_O} resulting in a DOS attack. Unauthorized modification of the encrypted object can be detected by means of a signature (generated using the sharer's private key) that can be stored within the puzzle Z_O.

C. Collusion Attacks

First, let us consider collusion between entities who do not know the context (e.g., SP, DH and users in $S_T - R_O$) and those who know it (e.g., users in R_O). In this case, those who know the context C_O (at least ζ_O correct answers), the encryption key K_O or the decrypted object O can trivially share these with others through a covert communication channel. Such a form of collusion is extremely difficult to protect against. Sharers can periodically modify the puzzle Z_O and/or the encryption key K_O (by re-encrypting the object) to partially protect against such collusion attacks.

Next, let us consider collusion between users who do not know the context C_O (i.e., users in $S_T - R_O$) and a malicious service provider SP who also does not know the context. More specifically, each user in $S_T - R_O$ may not know the context C_O completely, but they may partially know it, i.e., less than ζ_O correct answers. Then, a malicious SP can collude with a set of these users in $S_T - R_O$ and let them know through a covert channel the responses that verified correctly (despite the fact that each user would have less than ζ_O correct responses). On receiving the verification, this set of users could collaboratively determine a list of at least ζ_O correct answers, which can be then used to retrieve the object-specific secret M_O, and thus, the decryption key K_O. We assume a semi-honest SP that follows the protocol truthfully, and thus, our scheme is not secure against this extremely strong collusion scenario. Nevertheless, our scheme is secure against collusion among users in $S_T - R_O$, provided the service provider SP honestly executes the protocol. Our construction is also secure against collusion between SP and DH, provided they collectively do not know the context C_O and they do not collude with any users.

The above analyses for non-colluding SP and DH also holds for construction 2, especially in the semi-honest case. As the DH and SP only possess the perturbed access tree τ', the security of the cryptographic hash function will prevent efficient reconstruction of the original access tree τ without the knowledge of the context C_O. Moreover, we rely on the security guarantee of CP-ABE [19], which will prevent correct construction of the private key SK, and thus decryption of the ciphertext CT, without knowledge of the context C_O. Both SP and DH can act maliciously and achieve denial of service by manipulating the perturbed tree τ', public key PK, master key MK and the perturbed ciphertext CT'. Nevertheless, DOS attacks are beyond the scope of the current work. In the case when SP colludes with users in $S_T - R_O$, Construction 2 suffers from the same weakness as construction 1. However, similar to construction 1, it is secure against collusion between the SP and the DH, provided they both collectively do not know the context.

VII. Implementation

In this section, we outline implementation details of our constructions. We have implemented both constructions as third-party Facebook applications hosted on Amazon EC2 [20]. It should be noted that for demonstration purposes our application is hosted on a third-party provider such as Amazon EC2, rather than on an actual OSN provider. However, such an access control service can also be easily adopted (and hosted) by a popular OSN provider such as Facebook. Until then, existing Facebook users can take advantage of the proposed access control mechanisms by means of our publicly-available third-party application which can be hosted on a third-party provider of their choice.

Common features: Both implementations are interfaced with a *Facebook canvas application*. The sharer is required to grant permission to the application in order to post objects on Facebook. For simplicity, currently in our implementations the service provider SP and the storage service DH are located on the same server, but it can be easily extended to have both of them on physically separate servers. In order to provide confidentiality and authentication, all communications between users and our application on Amazon EC2 is carried over HTTPS.

A. Details of Implementation 1

The first implementation works across platforms and uses mostly JavaScript and HTML on the client end. All sharing and retrieving actions are performed in a JavaScript and cookie enabled web browser. Neither the sharer, nor the receiver, needs to install any additional supporting software. Our application enables users to use the access control functionality without leaving Facebook and offers a smooth and easy-to-follow interface. GibberishAES [21] is used for JavaScript-based symmetric encryption. All hash values are computed using SHA3 implementation of CryptoJS [22].

Sharing content: For sharing a new object, the application presents an HTML form (Fig. 5) to the sharer for inputting the object to be shared, context questions and corresponding answers. It requires the sharer to input the value of the threshold k and automatically detects the total number of contexts N by counting the number of question-answer pairs entered by the sharer. When the sharer submits this information, a JavaScript function is invoked to perform a number of client-side operations. This function computes a random secret M_O, corresponding hash K_O, and a puzzle specific key K_{Z_O}. Then, the object is encrypted using AES encryption with key K_O. Shamir's secret sharing algorithm is executed on M_O, and the obtained shares are XOR encoded with the context answers entered in the HTML form. The hash values of the answers concatenated with the puzzle-specific key K_{Z_O} are also computed. This completes the client-side computations for the sharer. The puzzle Z_O is then uploaded to the application server on Amazon EC2.

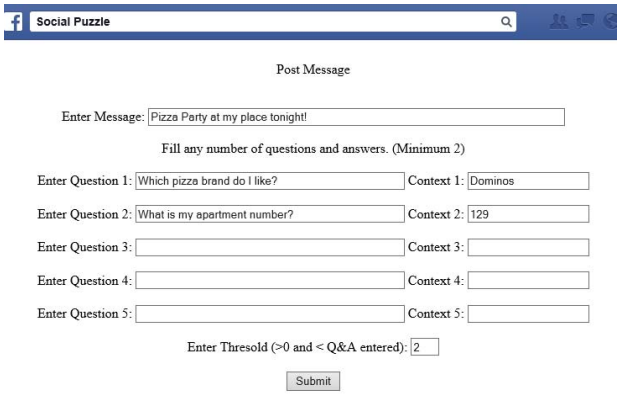


Figure 5. Impl. 1: HTML form with input boxes for message, associated contexts, corresponding questions and threshold k

The server component of the application maintains a MySQL database for storing information about all the puzzles. On receiving a new puzzle from a sharer, the server component adds a new entry in the MySQL *puzzle* table with a unique puzzle identifier. This identifier is then used to generate a hyperlink or URI which is posted on the sharer's Facebook profile (to the sharer's social network). The sharer can also choose to impose an additional layer of privacy control by means of Facebook's privacy settings (Fig. 6).

Receiving content: Sharer's friends (or receivers) who see the above post are expected to click on the hyperlink, if they wish to access the shared data object. This leads the receivers to an interface, where the server fetches the puzzle from the database and presents them with a randomized set of questions from the puzzle (Fig. 7). An HTML form to accept input from the receivers is also displayed. On receiving the answers to the questions from the receiver, a JavaScript subroutine (at the receiver) writes all the answers to a local cookie file. Another JavaScript function overwrites

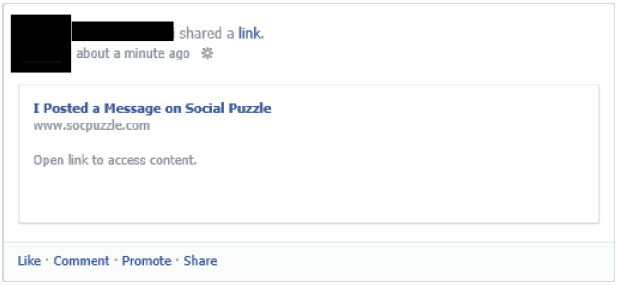


Figure 6. Impl. 1: A sample post made on Facebook

the answer fields in the HTML form with the corresponding hash values. These hashes are then sent to the server.

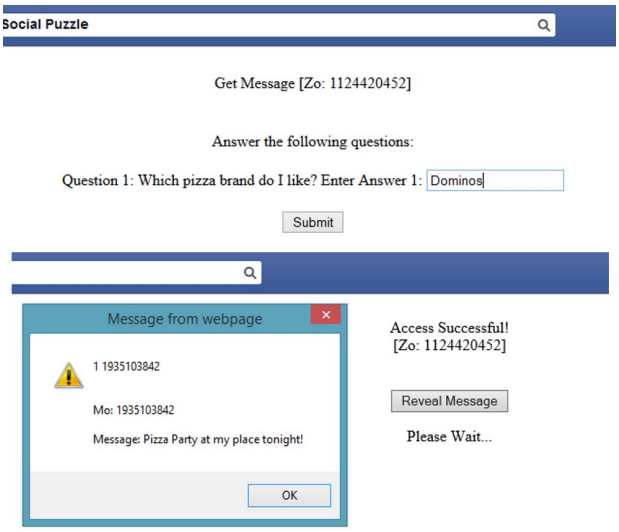


Figure 7. Impl. 1: Encryption key is reconstructed from puzzle answers and message is revealed

The server component of the application matches the hashed answers from the receiver to the hashed contexts stored in the database. If the threshold is not satisfied, the server displays an error message. If the threshold is satisfied, the server redirects the receiver to the encrypted object. The receiver also receives values of the shares encoded (XOR'ed) with the correct context answers. On receiving these encoded shares and the encrypted object, the receiver first retrieves the actual answers from the cookie file. Then, the answers are XORed back with the encoded shares to retrieve the original shares. The original random secret M_O is then computed from the shares using Lagrange polynomials. After calculating K_O from M_O, AES decryption is performed to reveal the encrypted object.

B. Details of Implementation 2

Our second implementation uses the publicly-available CP-ABE implementation. As the CP-ABE toolkit is currently available only for the Linux platform, this implementation is restricted only to those users (both sharers

and receivers) who operate a Linux system pre-installed with the CP-ABE toolkit. Moreover, as it is currently difficult to invoke CP-ABE library functions directly from the browser, users may have to switch between the browser and stand-alone components of the implementation in order to complete the access control functionality. This may create discontinuity in user-interaction flow while using the application. This could be addressed by developing a browser plugin that interacts with CP-ABE libraries from within the browser. Another issue that we encounter in this implementation is that the encoding of the access tree τ within the ciphertext CT is not known, thus preventing us from perturbing and reconstructing the access tree. To overcome this problem, we currently do not remove the original access tree τ from the ciphertext CT before storing it on the server. This action affects the surveillance resistance property (only in the implementation), but not the core access control functionality. These implementation shortcomings in our current version will be addressed in the future. Lastly, we compute all hash values in this implementation using SHA1 (available with OpenSSL [23]). For GUI, we use the Qt widget toolkit application framework [24].

Sharing content: In order to share a new object, the sharer executes a client-side Qt application. This application takes as input the object to be shared, associated context questions and corresponding answers, value for the number of contexts N, and value for the threshold k (Fig. 8). The object is stored in a file named *message.txt*. The values of N and k, the questions and the hashes of answers are written to another file *details.txt*. The *cpabe-setup* function is called in the background to generate master key file *master_key* and public key file *pub_key*. Then, the *cpabe-enc* process encrypts *message.txt* and replace it with *message.txt.cpabe*. The cURL [25] library is invoked to upload *details.txt*, *master_key*, *pub_key*, and *message.txt.cpabe* to the server component of the application running on the Amazon EC2 server. If all files are uploaded successfully, the server application assigns the puzzle a unique post (or puzzle) identifier. The server stores the hashes of all the context answers (along with the post identifier) in a database and deletes these hashes from *details.txt*. A reply is sent back to the client-side application containing the post identifier. The client-side application prompts the sharer to copy the post identifier and pass it to a Facebook canvas application (similar to the first construction). The Facebook application reads the post identifier using JavaScript, generates a hyperlink similar to the first implementation, and shares it on Facebook to the sharer's social network.

Receiving content: Sharer's friends (or receivers) who see the above hyperlink are expected to click on it, if they wish to access the shared data object. This leads them to the Facebook application, where the server displays the post identifier and prompts the receiver to copy the post identifier and pass it to a client-side Qt application for receivers. Once

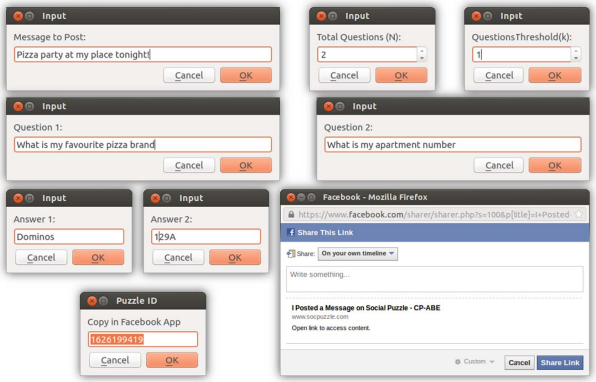


Figure 8. Impl. 2: Sequential interaction with sharer

the Qt application reads the post identifier, it downloads the corresponding *details.txt* file (with omitted hashed answers) from the server using the cURL library. The application then reads *details.txt* and presents prospective receivers with the questions from it. After the receiver answers the questions, the application computes the hash values of answers and sends them back to the server for verification. If less than k hashed answers matches, an error message will be returned. If verification succeeds, the server gives access to *message.txt.cpabe*, *master_key*, and *pub_key* files. The Qt application downloads these three files using cURL. The application inputs the earlier user entered answers to the *cpabe-keygen* function in order to generate the decryption key file *my_priv_key*. The *my_priv_key*, *master_key*, and *pub_key* files are used to decrypt *message.txt.cpabe* by using the *cpabe-dec* function. Finally, the Qt application displays the contents of *message.txt* to the receiver (Fig. 9).

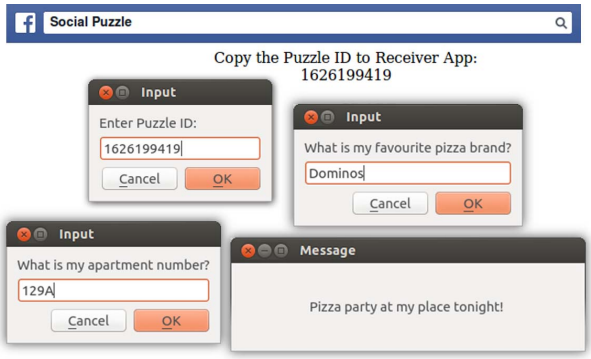


Figure 9. Impl. 2: Receiver solving a puzzle and reading the message

VIII. Evaluation

In this section, we present an evaluation of preliminary performance-related measurements that we obtained by executing our applications in a controlled setting as well as a discussion of usability issues related to our proposed solution.

Experimental setup: For both implementations, we use a common PC hardware comprising of a quad core 2.5 GHz CPU, 1GB RAM and a 802.11n WLAN interface operating at 60 Mbps. The system is running a Ubuntu version 13.04 OS. Additionally, performance of the first implementation was assessed on a Nexus 7 tablet, and compared with the performance on the PC. Latest versions of Firefox browser were used on both devices. The second implementation could not be benchmarked on the tablet because of its Linux dependency. Experiments were performed for message lengths of 100 characters, answers of 20 characters and questions of 50 characters long. Measurements were taken for varying number (N) of contexts, while the threshold k is set to 1. As CP-ABE does not support (1,1) threshold, observations start from $N = 2$. We do not include user interaction time in our observations.

Implementation 1 vs. Implementation 2 on PC: Figure 10(a) and 10(b) shows the breakdown of the *local processing delay* and *network delay* (including server-side processing) for the sharer and the receiver, respectively. The network delay of Implementation 2 (I2) is worst as compared to Implementation 1 (I1). For each upload by the sharer in I2, the cURL library is used to upload four different CP-ABE related files (total $\sim$600KB in size) to the server. The network delay observed for I2 (Figure 10(a)) is both, due to this considerably large amount of data being transferred between the client and the server, as well as, due to the additional overhead caused by the cURL library. The instability in the measurements, which mostly shows an increased delay with increasing context size, seems to be due to the unpredictability of the communication network speed. Also, I2 has slightly higher local processing delay because of the greater computational complexity of CP-ABE. The combined delay in I1 is extremely low for both sharer and receiver, while for I2 it is noticeably high at the sharer and comparatively lower at the receivers.

PC vs. Tablet for Implementation 1: Figure 10(c) and 10(d) shows the breakdown of local processing delay and network delay for sharer and receiver, respectively. We observe that I1 performs better on PC than on tablet. However, the overheads are insignificantly low on both devices.

Usability Aspects: By means of the implemented Facebook-compliant prototypes, our goal was to validate the effectiveness and efficiency of the proposed access control mechanisms. We observed from our experimental evaluations that users can successfully and efficiently share data on the Facebook OSN using our prototype applications. However, currently we have no evidence on of how intuitive the idea of context-based sharing is to OSN users and what features OSN users expect in such an application. As with any online service, it is vital to understand the related usability aspects of the proposed paradigm in order to improve its practical feasibility, i.e., who, where and how such applications will be used. To evaluate application usability, feedback gathering activities such as focus groups, surveys, user-experience studies and ergonomic assessments can be conducted. The ISO standard 9241 Part 11 [26] provides specific guidelines for evaluating applications involving human-computer interactions with respect to the goals of effectiveness, efficiency and satisfaction. In order to improve the usability of our applications, we intend to conduct an on-campus survey and user-study by following the guidelines set forth in the ISO standard 9241. We also plan to add additional features to our applications, e.g., support for non-textual data, picture-based puzzles and automated client-side context recommendations, to improve its ease-of-usage and to enhance user-experience.

IX. Conclusions

In this paper, we proposed and implemented two novel context-aware access control mechanisms which empowers users to regulate access to their shared data in OSN services. Instead of controlling access to shared data based on users or user-attributes, the proposed mechanisms focus on controlling access to data based on the knowledge of the context associated with the data. By means of the proposed mechanisms, OSN users can not only enable fine-grained access control and improve relevance of the shared data, but also protect it against surveillance from curious service providers. We analyzed the security of the proposed mechanisms under various passive and active adversarial scenarios. We also verified the correctness and performance of our implementations by means of empirical evaluations.

Acknowledgment

The authors would like to thank all the anonymous reviewers for their insightful comments and suggestions.

References

[1] G. A. Fowler, "Facebook: One Billion and Counting," *The Wall Street Journal*, October 2012.

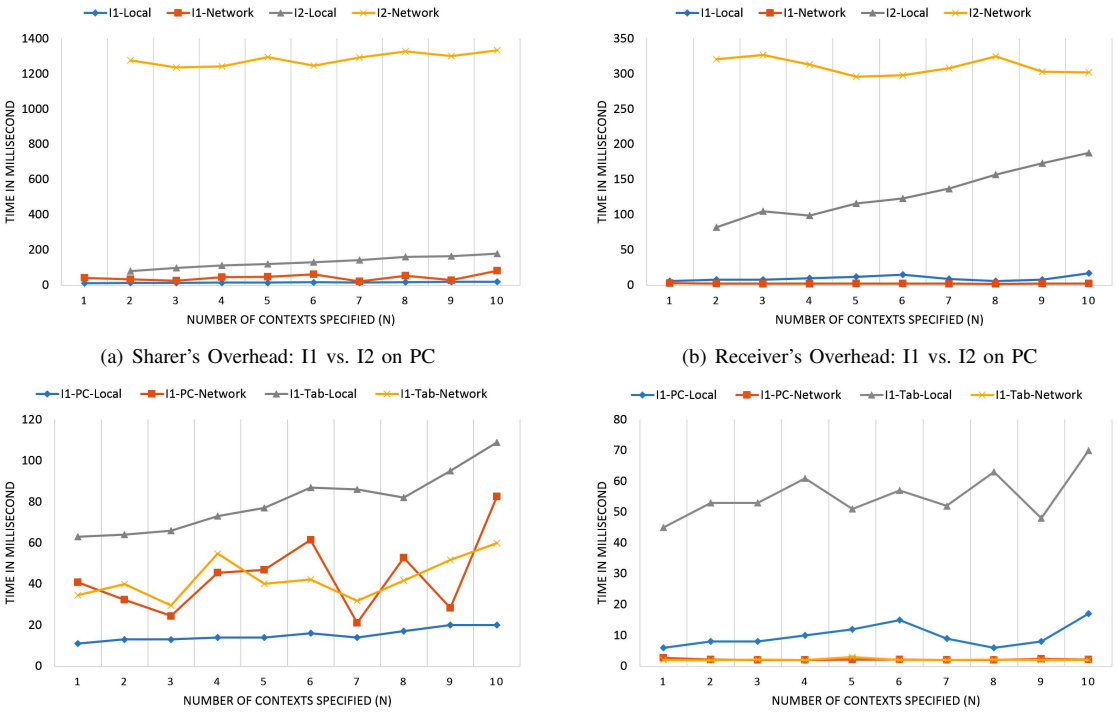
[2] R. Gross and A. Acquisti, "Information revelation and privacy in online social networks," in *Proceedings of WPES '05*, 2005.

[3] "Facebook privacy settings." [Online]. Available: https://www.facebook.com/settings?tab=privacy

[4] M. Johnson, S. Egelman, and S. M. Bellovin, "Facebook and privacy: It's complicated," in *Proceedings of SOUPS '12*, 2012.

[5] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman, "Lockr: better privacy for social networks," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, 2009.

[6] W. Luo, Q. Xie, and U. Hengartner, "Facecloak: An architecture for user privacy on social networking sites," in *Proceedings of IEEE CSE '09*, 2009.



(a) Sharer's Overhead: I1 vs. I2 on PC

(b) Receiver's Overhead: I1 vs. I2 on PC

(c) Sharer's Overhead: PC vs. Tablet for I1

(d) Receiver's Overhead: PC vs. Tablet for I1

Figure 10. Performance Measurements

[7] M. M. Lucas and N. Borisov, "Flybynight: mitigating the privacy risks of social networking," in *Proceedings of WPES'08*, 2008.

[8] V. Pouli, J. S. Baras, and A. Arvanitis, "Increasing message relevance in social networks via context-based routing," in *Proceedings of MSM workshop*, 2012.

[9] P. Jagtap, A. Joshi, T. Finin, and L. Zavala, "Preserving privacy in context-aware systems," in *Proceedings of ICSC '11*, 2011.

[10] M. Dürr, F. Gschwandtner, C. K. Schindhelm, and M. Duchon, "Secure and privacy-preserving cross-layer advertising of location-based social network services," in *Mobile Computing, Applications, and Services*, 2012.

[11] J. Li, Y. Tang, C. Mao, H. Lai, and J. Zhu, "Role based access control for social network sites," in *Proceedings of IEEE JCPC'09*, 2009.

[12] S. Jahid, P. Mittal, and N. Borisov, "Easier: Encryption-based access control in social networks with efficient revocation," in *Proceedings of ACM ASIACCS'11*, 2011.

[13] C.-m. A. Yeung, I. Liccardi, K. Lu, O. Seneviratne, and T. Berners-Lee, "Decentralization: The future of online social networking," in *W3C Workshop on the Future of Social Networking Position Papers*, 2009.

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

[15] ——, "Scramble! your social network data," in *Proceedings of PETS'11*, 2011.

[16] F. Beato, I. Ion, S. Čapkun, B. Preneel, and M. Langheinrich, "For some eyes only: protecting online information sharing," in *Proceedings of ACM CODASPY'13*, 2013.

[17] B. Carminati, E. Ferrari, and A. Perego, "Enforcing access control in web-based social networks," *ACM TISSEC*, 2009.

[18] A. Shamir, "How to share a secret," *Comm. of ACM*, 1979.

[19] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-Policy Attribute-Based Encryption," in *IEEE S & P '07*, 2007.

[20] "Amazon EC2." [Online]. Available: http://aws.amazon.com/ec2

[21] M. Percival, "GibberishAES." [Online]. Available: https://github.com/mdp/gibberish-aes

[22] "CryptoJS." [Online]. Available: https://code.google.com/p/crypto-js

[23] "OpenSSL." [Online]. Available: http://www.openssl.org

[24] "Qt project." [Online]. Available: http://qt-project.org

[25] "cURL." [Online]. Available: http://curl.haxx.se

[26] *ISO 9241-11:1998 - Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11: Guidance on usability*.