Policy-Hiding Cryptographic Schemes in Online Social Networks

#1 Perugu Chandrakala , *2 Hajarathaiah Koduru , #3 K.Koteswararao

1M.Tech Student, Computer Science & Engineering, Narasaraopeta Engineering College, Narasaraopet, Guntur, Ap, India.
2,3 Assistant Professor, Computer Science & Engineering, Narasaraopeta Engineering College,
1 chandrakala.perugu@gmail.com
2 azaroop@gmail.com,
3 koteshk999@gmail.com

***Abstract*—Privacy concerns in online social networking services have prompted a number of proposals for decentralized online social networks (DOSN) that remove the central provider and aim at giving the users control over their data and who can access it. This is usually done by cryptographic means. Existing DOSNs use cryptographic primitives that hide the data but reveal the access policies. At the same time, there are privacy-preserving variants of these cryptographic primitives that do not reveal access policies. They are, however, not suitable for usage in the DOSN context because of performance or storage constraints. A DOSN needs to achieve both privacy and performance to be useful. We analyse predicate encryption (PE) and adapt it to the DOSN context. We propose a univariate polynomial construction for access policies in PE that drastically increases performance of the scheme but leaks some part of the access policy to users with access rights. We utilize Bloom filters as a means of decreasing decryption time and indicate objects that can be decrypted by a particular user. We evaluate the performance of the adapted scheme in the concrete scenario of a news feed. Our PE scheme is best suited for encrypting for groups or small sets of separate identities.**

I.INTRODUCTION

Centralized online social networks collect and store private information and are thus prone to privacy leaks, ranging from modest data mining for advertisement purposes to direct transfer of data to third parties (e.g. recent reports on National Security Agency’s global surveillance program [1]).

One of the alternatives is a decentralized provider-less P2P architecture (DOSN), in which users have control over their data and who can access it. This relies on untrusted storage and thus requires cryptographic means to protect data. General-purpose cryptographic profiles in DOSN typically use either attribute-based encryption (ABE)[2], broadcast encryption (BE) [3], or symmetric encryption. The main aim is to protect confidentiality (of the content itself) and privacy (of information about the content or access policies). While all of the existing systems provide confidentiality guarantees, privacy is usually less addressed.

An access control mechanism should be privacy preserving. By privacy preserving we mean that the user should be able to decrypt only the objects for which he satisfies the access policy; encrypted objects should not to reveal users who have access to these objects; the quantity, size, and type of objects should be unknown to the user unless he can decrypt them.

The choice of cryptographic primitive does not only involve performance and privacy as parameters, but also the expressiveness and semantics of access policies (easily) supported. In ABE, a key is associated with one or more attributes, and a ciphertext is encrypted for some policy of attributes (e.g., “friends or family”). In BE, ciphertexts are instead directed to sets of recipients. This difference in the cryptographic primitive typically also affects the semantics of the system when one e.g. adds a new friend. By default, in ABE the new friend would be able to access all previous content addressed to friends, while in BE she would not. A drawback of both BE and ABE is that the standard implementations reveal the access polic specified together with each ciphertext (as the access policy is needed for decryption).

While current DOSNs are not privacy-preserving as outlined above (see Section II), ABE-based DOSN systems could be patched to provide privacy by using a privacy-preserving version of this cryptographic primitive [4], but efficiency would be lost because of the quadratic growth of the ciphertext size in the number of attributes. BE-based systems could use anonymous BE, ANOBE [5], but then long ciphertexts make it inefficient.

Computational efficiency and storage efficiency are crucial for DOSNs which are characterized by a large number ofusers and objects, and by the absence of a centralized storage. Privacy preservation, being the main motivation behind DOSN, is of equal importance.

Striving to achieve privacy, without sacrificing performance, we introduce predicate encryption (PE) to DOSN. Like ABE, PE uses attribute-based policies. When creating a ciphertext, the encryptor specifies an access policy and only those users whose keys satisfy the policy can decrypt. We chose PE because it does not reveal access policies. It cannot, however, be used directly for DOSN because of its modest performance.

In current PE schemes, ciphertext size, encryption and decryption times are linear in the size of vectors associated with decryption keys and ciphertexts. Standard constructions result in very large vectors already for mildly complex access policies. We propose a univariate polynomial construction for access policies that has a short vector and thus drastically increases performance of the scheme. It provides full privacy preservation from external observers and partial from users who have access according to this policy (they can infer some 2014 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops) part of the access policy). Hiding access policies in PE also prevents legitimate users from knowing whether they will be able to decrypt a ciphertext. We address this by using Bloom filters.

For the purpose of this paper, we have the following system model of a DOSN. We assume the existence of distributed storage for storing data. The storage is untrusted, consequently the access control mechanism is independent of storage, and is based on encryption. Every encrypted or unencrypted object stored in the profile is world-readable. Users set access policies for their data and a particular cryptographic scheme is used to realize the access policy the user decides on. The profile owner creates personal decryption keys for each of his friends. These keys are stored in a “key file” (one for each friend) on the profile of the profile owner encrypted under the per-friendship symmetric key shared by the profile owner and the friend. Thus, we require that during friendship establishment users create shared secret keys that are used to retrieve keys for decrypting data on each other’s profile. An object is encrypted under some access policy. Only the user whose decryption key satisfies the access policy of an object can decrypt the object. A digital signature scheme (independent from the encryption scheme) is used for message authentication purposes. Each user is assumed to have a private key for signing messages that he posts.

A. Our contribution

We introduce a PE scheme in the DOSN setting and develop modifications to efficiently adapt it to this setting. We evaluate the performance of the modified scheme. We also present a performance-friendly privacy-preserving mechanism: Bloom filters for indication of objects that a user can decrypt.

B. Paper outline

We start with discussing related work in Section II and continue with a description of a proposed cryptographic primitive and our modifications in Section III. Section IV is devoted to the evaluation of efficiency of our construction. Section V concludes the paper.

II. Related Work

There has been a lot of research on decentralized social networks [6], [7], [8], [9]. The biggest concerns are distributed storage, the access control mechanism, and privacy. An early version of the PeerSoN [10], [11] P2P social network used a DHT to look up data and a combination of symmetric and asymmetric cryptography for encryption-based access control on untrusted storage. Privacy was not sufficiently addressed since userIDs (or public keys) were stored alongside encrypted data. Safebook [6] uses trusted friends to store data on their computers and to ensure privacy. Confidentiality is again achieved with a combination of symmetric and asymmetric encryption, and a DHT is used as a lookup service to find a path to the stored data. The privacy of the scheme is partial because explicitly trusted parties (most trusted friends that serve as mirrors) can trace communication parties, but communication privacy is protected from external observers via multi-hop routing. Persona [7] relies on untrusted storage and uses ciphertext policy attribute-based encryption for access control. To provide specific rights to stored objects, the profile owner defines access control lists (ACLs) and the storage enforces them. This scheme, however, does not guarantee privacy. ACLs contain the users’ public keys and their access rights. The storage authenticates the users and authorizes their actions based on the entries in the ACL. Cachet [8] is an update of the Decent architecture [12]. It uses a DHT to store data and uses ABE to ensure confidentiality. In the ABE used the access policy is described openly in the header of the ciphertext. The authors observe the resulting privacy violation, but only address it partially by hiding these headers from the storage system. Users can still observe headers and thus can see plaintext ABE access policies. For efficiency, the authors used caching of information and store the unencrypted version of this information on the nodes that satisfy the ABE policy (nodes that are able to decrypt). Thus users will know for whom the content is encrypted, and they may even be able to trace the requests of other people who also can decrypt the same information. G¨unther et. al. [9] describe two solutions for publishing of content on social network profiles. One solution uses broadcast encryption with pseudonyms. Pseudonyms are needed to provide privacy protection and patch the used BE scheme which leaks the set of recipients. Pseudonyms give a limited anonymity property [9], but it is still possible to see which pseudonym is authorized to decrypt what. Taking into account that other users might have some additional information about an event/question that the encrypted message covers, we argue that the protection is not sufficient as users may infer the identity behind the pseudonym. Their second construction is based on symmetric encryption. It requires for each attribute value pair in the system and for each user from the set of recipients of that value to have a separate decryption key. This approach scales poorly to large system sizes. Tahoe [13], a distributed file system, uses symmetric encryption. Each encrypted file is associated with at least two unique cryptographic values/capabilities. One is a symmetric encryption key and the other one is a hash value for checking integrity. To give access to an encrypted file to a user, one has to share these two cryptographic capabilities with this user. Taking into account the large number of friends in social networks, such sharing results in too much overhead and becomes prohibitively expensive. By grouping a set of files into a directory (a file that contains all cryptographic capabilities required to read/write any file from the set [13]) and then sharing cryptographic capabilities only for this directory we could partially solve the problem, but then we lose flexibility of fine-grained access to files. Anderson et. al [14] describe a social network that divides a user profile in discrete encrypted blocks. Symmetric secret keys for these blocks are shared between users who should have access to information stored in these blocks by using hierarchical group key management schemes. We argue, that it is not obvious that there exists a hierarchy of users/groups (unlike the hierarchy of files and directories) in a profile of an average user besides the most simple one, in which any group is a subset of group "friends" containing all connections of the profile owner. In a system without access rights hierarchy, a hierarchical group key management scheme will perform no better than a simple system based on shared keys for groups. Besides, to give a different view of a folder/directory (e.g. the "wall") to different users while keeping cryptographic overhead at bay would require elaborate constructions exceeding the possibilities of the hierarchical group key management scheme. Our system seamlessly supports this. Our approach is to use a PE scheme that hides access policies and is thus more privacy preserving than the related work. This scheme needs adaptations to achieve high performance and functionality in the DOSN context. We describe them in the following.

III. PREDICATE ENCRYPTION

Predicate encryption (PE) [15] is a cryptographic primitive that provides access control of encrypted data using attribute based policies. When creating a ciphertext, the encryptor specifies an access policy and only those users whose keys satisfy the policy can decrypt. The decryption keys are generated by the encryptor using a master secret.

The functionality of PE schemes is similar to that of ABE, but the policies supported by PE schemes are in fact less suitable for our goals than those of ABE. The reason we pursue PE schemes is that in many PE schemes ([15], [16], [17]) nothing about the policy is revealed from the cipher text. This property is called "data privacy" [18] and it is in line with our privacy requirements defined in Section I. We do not support a so called "predicate privacy" property [18], when a decryption key does not reveal any information about access rights associated with it. We argue that in the DOSN setting a user possessing a decryption key and knowing some background social information (about a person, an event, etc.) can eventually infer access rights associated with the key in his possession by seeing what he can decrypt.

The expressiveness of access policies depends on the chosen PE scheme. The most expressive class of PE schemes [15], [16], [17] uses so-called inner-product predicates. In these, each key is associated with a vector ¯v of length ` and the ciphertext is associated with a vector ¯a of equal length over Z_N (for some large integer N). Decryption succeeds only

If $\sum_{i=1}^{\ell} v_i a_i = 0 \mod N$.

In other words, one can decrypt a ciphertext when a vector associated with the ciphertext is orthogonal to a vector associated with the decryption key. One of these vectors is an access policy and another one represents an attribute.

In our case, it is the profile owner who issues decryption keys to his friends. Friends cannot see or modify vectors associated with their decryption keys.

We remark that while we describe PE as having attributes associated with keys and an access policy associated with cipher texts, the dual is also prevalent in the literature. With our focus on inner-product PE schemes, the difference between what is a policy and what is an attribute is not important as the inner-product is a commutative operation.

A. Inner-product access policies

Based on the primitive of vector inner products, one can implement various more complex access formulas. The construction of these go via polynomial evaluation, following a construction by Katz et al. [15]. Here, the access policy is represented by a polynomial $p(x) = \beta_d x^d + \cdots + \beta_0 x^0, \beta_i \in \mathbb{Z}_N$ and decryption is possible if the key corresponds to one of the roots of the polynomial.

To implement polynomial evaluation, we represent the access policy by the vector $\bar{p} = (\beta_d, \cdots, \beta_0)$, and a key associated with an attribute A as $\bar{A} = (A^d, \ldots, A^0)$. The inner product of two such vectors corresponds to evaluating the polynomial p(A). As these vectors must be of the same length, the degree of the polynomial (and thus the complexity of the access policy which can be expressed) must be decided when the keys are issued. We remark that in the actual construction, both these vectors are "blinded" by multiplying them with a random element
$\gamma \in \mathbb{Z}_N$, i.e., $\bar{p} = \gamma(\beta_d, \cdots, \beta_0) = (\gamma\beta_d, \cdots, \gamma\beta_0)$. This provides randomization of ciphertexts such that multiple ciphertexts with the same policy still look different. For clarity of exposition, we omit this randomization from the notation. Based on polynomial evaluation, we can in principle support Boolean CNF or DNF formulas. Firstly, for disjunction, the policy "$A \vee B$" can be expressed using the following univariate polynomial:

$p(x) = (x - A)(x - B) = x^2 - (A + B)x + AB$

The resulting vector is thus $\bar{p} = (1, -(A + B), AB)$ and keys representing either A or B can decrypt the corresponding ciphertext.

In our setting, a user has several attributes (e.g., Alice is both a friend of Bob's as well as in the same programming club as he is). Ideally, we would like to give each user a single key representing all of her attributes. The natural solution to this is to go to multivariate polynomials, and the evaluation mechanism proposed earlier translates straightforwardly. However, this approach cannot be directly applied in our setting and we now proceed to show it and explain why.

We now naively try to represent the access policy "$A \vee B$" and express it with a multivariate polynomial:
$p(x_1, x_2) = (x_1 - A)(x_2 - B) = x_1x_2 - Bx_1 - Ax_2 + AB$

which we translate to a vector $\bar{p} = (1, -B, -A, AB)$. A key with attributes $x_1 = A$ and $x_2 = C$ would be represented by the vector (AC, A,C, 1). Here we spot the first problem with this construction: attributes are now tied to specific variables of the polynomial! Thus, a key with $x_1 = C$ and $x_2 = A$ cannot decrypt the ciphertext as it is not orthogonal to the policy. This issue can be alleviated by making the access policy large, e.g. specifying the policy "$x_1 = A \vee x_2 = A \vee x_1 = B \vee x_2 = B$". This rapidly yields polynomials of high degree, and as lengths are fixed when keys are issued, we would need to pay a high overhead on every ciphertext. Performance of these schemes is linear in the vector length with a relatively high constant. We evaluate performance in Section IV, our finding there is that a vector length of about 10 elements still yields good performance.

Other Boolean constructions such as conjunction and negation can also be supported, with the same issues as above when multinomial are used (i.e., the atomic clauses are of the form $x_i = a_i$, while we need "has attribute a_i"). We omit details here and instead proceed to our simplified construction.

B. Our construction for access policies

Based on the observation that univariate polynomials result in more efficient schemes than multivariate ones, we propose to construct access policies solely on univariate polynomials. To support conjunctions we introduce virtual attributes, and transform all policies to a disjunctive normal form (DNF).

We will be referring to the PE scheme with our construction for access policies as "PE with modified access policies" (PEMAP) in the rest of the article.

This construction builds on the polynomial construction previously described, where we saw that a polynomial of degree `+1 can represent a disjunction over ` attributes. Here, a user receives one key for each attribute she has, and will then have to use the appropriate one to decrypt (we show how she learns which key to use in Section III-C). By itself this would support only disjunctive access policies (i.e., "friend OR family OR . . . ").

To deal with conjunctions we simply add a new virtual attribute for each conjunction of attributes. While this creates an exponential increase in attributes (and thus keys issued to users), the approach is feasible as long as no user is placed in too many groups by the profile owner. In practice most people only use a very small number of groups 6 - 13 ([19], [20]) for access control in OSNs. We have not found data on how many groups a single user is typically placed in, but given the low number of groups used, we speculate that a given user would very rarely be in a large fraction of the groups used, and that the scheme would thus work in practice, despite the exponential scaling.

With this design, we may represent (A^B) by the virtual attribute M, and (B ^ C) is mapped to some different virtual attribute, e.g. V . Thus, if a user is a member of groups A and B, she would also be given a key corresponding to M. In general, a user who is a member of g groups will be given $2^g - 1$ decryption keys. We further add an identity attribute ID_i, which is an individual attribute for each user. This allows us to name individual users in our access policies. Thus, in total in our scheme, a user will receive 2^g keys. For example, a user who is a member of 4 groups of the profile profile owner will receive 2^4 keys. As we mentioned in the previous paragraph, a user would very rarely be in a large number of groups. Assume we want to encrypt for "$A \vee (B \wedge C)$". This policy is expressed as the following univariate polynomial:

$$p(x) = (x - A)(x - B \wedge C) = (x - A)(x - V)$$

The corresponding vector is $\bar{p} = (1, -(A + V), AV)$.

We have a trade-off between efficiency and functionality of the scheme: the longer the vector, the larger the clauses that can be used in formulas, but the higher the overhead. Recall that the vector length is fixed once keys has been issued. This vector length is a profile-wide parameter that every profile owner can choose. The choice affects the sizes of ciphertexts, and decryption keys, as well as the computational overhead of the scheme. It, however, neither affects the number of groups that the profile owner can create (this number is unbounded), nor the number of keys that some user has. We suggest using vectors of length 10, based on evaluation where we found that length to yield adequate performance of 70 ms decryption time (see Section IV) while supporting a reasonable complexity of access formulas. With this vector length an access formula can support up to 9 clauses.

To make inner-product PE schemes efficient enough for our application we use a construction that require multiple decryption keys per user. However, unfortunately this gives rise to some privacy issues, which was what we set out to address. We believe these are still significantly milder than those of ABE where the whole access policy is leaked. Specifically, the problems we note are that by keeping track of which of his keys were able to decrypt what ciphertexts, a user may over time from context understand the meaning of his keys (e.g., if all ciphertexts accessed via key k_1 seem to be about the programming club, the user may infer that this key likely corresponds to that attribute). In addition, by simply counting the number of keys he receives, a user learns the number of groups he is a member of. The latter problem can be mitigated by padding: giving each user some constant number of keys where some of them are random and never used (but then over time, the user may identify these keys as being padding as they are never used).

Example: Let's set the vector length to 10 for all messages in our profile. Our friend who is a member of one group A will have two decryption keys: one for the group attribute and another one for the identity. The vectors will be:

$$\bar{k}_A = (A^9, A^8, \ldots, A^0)$$

$$\bar{k}_{ID1} = (ID1^9, ID1^8, \ldots, ID1^0)$$

Let's encrypt for: $A \vee B \vee D \vee (C \wedge E) \vee ID2 \vee ID3 \vee ID5$, where A,B,C,D, and E are groups and ID2, ID3, ID4 are identities of some users. The polynomial will have the following form:

$$p(x) = (x - A)(x - B)(x - D)(x - Q)(x - ID2)$$
$$(x - ID3)(x - ID5)(x - R1)(x - R2)$$

, where Q is a virtual attribute that represents C ^ E, Ri are random values used for padding. Ri should not be equal to any other group/identity attribute in our profile. By expanding this polynomial and taking coefficients as vector elements, we get the following vector:

$\bar{ct} = (1, -(A + B + D + Q + ID2 + ID3 + ID5 + R1 + R2), \ldots, -A \cdot B \cdot D \cdot Q \cdot ID2 \cdot ID3 \cdot ID5 \cdot R1 \cdot R2)$

Vectors $\bar{ct}$ and $\bar{k}_A$ are orthogonal, and thus our friend can decrypt.

C. Bloom filters

A profile in the DOSN contains multiple objects encrypted for different users. It is impossible for a user to determine if an object is encrypted for him without trying to decrypt it since the ciphertexts do not reveal access policies. The use could use a trial-and-error approach (sequentially trying to decrypt objects) for rendering the profile, but this becomes prohibitively expensive with the large number of objects. Therefore, we utilize Bloom filters [21] to speed up rendering and to show users in a privacy-preserving manner whether they can decrypt objects.

A Bloom filter is a space-efficient data structure that represents membership of elements in the set. From the data storage point of view, it is a bit array of some bounded size m. Bloom filters have two operations: add(x) and query(x), where x is some element. The add operation consists of hashing an element with several hash functions $h_1, \ldots, h_k$, which uniformly map the element to a number $h_i(x) = y_i$ 2 [1;m], and setting bits y_i to 1 in the array (initially the array is filled with zeroes). The query operation repeats the same hashing procedure, and then checks if the appropriate bits are set to 1.

In our construction, each key has an associated key ID, a random 128-bit string stored alongside the key, picked by the profile owner and known only to the profile owner and a user who has this key. These keyIDs are the values which are inserted and tested for in our Bloom filters. Two privacy problems need to be addressed when constructing the Bloom filters. Firstly, if an adversary learns which keyIDs are present in the Bloom filter, these will serve as pseudonyms for the users having access, which is a privacy violation. To remedy this, we add a nonce S to the Bloom filter and instead of inserting the keyID directly, we transform the input as: input = H(S||keyID) where H is a cryptographically strong hash function, e.g. SHA-256. The nonce S is stored alongside an encrypted object, see a format of the encrypted object in the end of this section.

The second issue is that the number of bits which are set to 1 in the Bloom filter allows an adversary to learn the approximate size of the recipient set. We address this issue via padding. Each profile owner will pick some fixed size greater than the number of his friends and ensure that the Bloom filters she publishes always contain that many values. To do so, after inserting the keyIDs describing the access policy, the profile owner samples random values and inserts them into the Bloom filter until it is sufficiently full. The price we pay for this padding is that Bloom filters need to be larger than they would be otherwise in order to maintain a reasonable false positive ratio.

With a Bloom filter, an encrypted object has the following format:

$$CT, PE_{enc}(PK,K,AP), S, BF(H(S\|kID_1), \ldots, H(S\|kID_n), R_1, \ldots R_{p\_n}), SGN \quad (1)$$

where CT - is an object encrypted with a random key K, K - a symmetric key used for AES encryption, PK- public key of the PE scheme used for encryption, AP - access policy, S - nonce, BF - Bloom filter, H - cryptographically secure hash function, kID_i - an identifier of a decryption key belonging to recipient i, R_i - random values for padding, p - padding size, SGN - signature of all the previous fields.

IV. Performance Evaluation

We have developed a lightweight simulator to estimate the efficiency of the proposed encryption-based access control. All tests described in this paragraph were run on the 2,67GHz

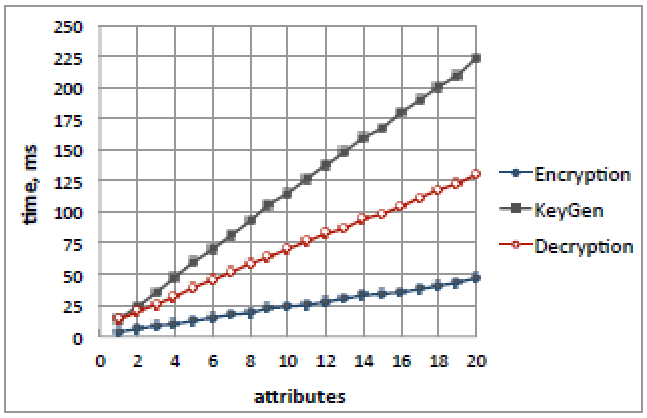

Fig. 1. PE scheme performance

Intel Core i5 M580 processor on the Ubuntu 12.10 machine. Measurements and estimation were done for a 128-bit security level and a 128-bit plaintext (the size of the symmetric key). We used a MIRACL cryptographic library [22] for the predicate encryption and OpenSSL v 1.0.1c for the underlying symmetric encryption (AES). All measurements were done with disabled multi-threading/multi-core.

The performance of AES was estimated using the OpenSSL benchmark. The AES-128 cipher in CTR mode showed 2400 MB/s processing rate. The time to create a random 128-bit AES key is negligible.

A. Pe

In this paper, we are using Park's PE scheme [16], where a cipher text contains (4n + 3) group elements of an elliptic curve with a bilinear pairing. The size of representing a group element varies a bit with the group selection, e.g. for a 128-bit security level NIST recommendations [23] give a range of 256-383 bits per element. The library we used for performance measurements used a curve with a representation of 360 bits. Thus, at the 128 bit security level with 128-bit plaintext the ciphertext requires:

PE.CT.size(l) = 102 + 180l bytes

where l is a dimension of the attribute vector. For 10 elements it is 1902 bytes.

We would like to emphasize that there are more space efficient schemes. A PE scheme by Okamoto et al. [17] requires only (n + 5) elements for the ciphertext, and thus OkamotoPE.CT.size(10) = 15 · 256 = 460 bytes.
The Figure 1 shows the performance of Park's PE scheme [16] with a Barreto-Naehrig curve (BN-128), without the AES component. We would like to point out that it is possible to achieve several times higher encryption/decryption rates for this scheme [24].

B. Bloom filter

The storage overhead imposed by Bloom filters should be as small as possible. The number of bits m needed to encode membership information in the Bloom filter depends on the false-positive probability p and the number of elements n to be inserted. The space efficiency is calculated according to the following formula [25]: $m = -n \ln p \ (\ln 2)_2$, while the optimal number of hash functions k for a given space efficiency level is $k = {}_{mn} ⇤ \ln 2$. For example, the size of the Bloom filter for 500 users with a false positive probability of 0,01 is 600 bytes.

The performance of a Bloom filter is determined by the performance of the underlying hash functions. Kirsch and Mitzenmacher [26] show that only two different hash functions $h_1(x)$ and $h_2(x)$ are required to implement a Bloom filter. From these, k hash functions can be simulated: $g_i(x) = h_1(x)+i \cdot h_2(x)$, where i 2 [0, k −1]. Therefore, an insertion operation requires h = 2n hashing operation, where n is a number of objects to be inserted. With modern hash functions such as CityHash or MurmurHash the time to compute these hash functions is negligible.

However, for privacy reasons, we also require a cryptographically strong hash operation prior to insertion into the Bloom filter. To measure the computational cost of this hashing operation, we used the OpenSSL benchmark function for SHA- 256. The time required to hash 500 blocks of 256 bits is below 1 ms.

C. News feed assembly

We saw in Section IV-A that the computational cost of PE is relatively high. To estimate the impact this would have in a real system, and to assist in the recommendation on vector length we considered the common OSN workload of rendering a news feed. For this purpose, we developed a news feed assembly simulation which assumes the existence of a DOSN based on a DHT for peer lookup and distributed storage for storing persistent data.

We use a simple pull strategy (updates are pulled from the friends) to assemble a one-day news feed from 300 friends. An average Facebook user makes approximately 10 wall posts per month [27], or nearly 100 per year [28]. For each friend profile we set a uniformly distributed random number of encrypted objects (news) in the interval [0; 2]. So, the total mean number of encrypted objects is 300, which is 3 times higher than the average per day value for 300 friends. In this simulation the user has access to all posts considered, so the Bloom filter false positive rate does not affect performance. We show how performance is affected by cryptographic overhead; the computational overhead for membership tests in Bloom filters is negligible.

We assume that our DHT operations (writes, reads) have the cost of Mainline BitTorrent DHT lookups with NR128- A routing strategy [29]. Lookups are run in parallel. We use inverse transform sampling from raw CDF data describing NR128-A [29] for our lookup time pseudo-random generator. The number of consequent DHT lookups needed to retrieve a friend's wall file depends on the DOSN architecture. We assume that a user needs one lookup to get the "root" file for the friend's profile, one for the file with names of files that contain decryption keys for all friends of the profile owner, one to get the file with decryption keys exactly for this user, and one for the wall file, thus 4 in total. Then each object requires one lookup. Thus the mean total number of DHT lookups for 300 objects is 1500. A mean DHT lookup time is 183 ms.

Figure 2 shows the dependency between the news feed assembly time, the decryption speed, and the number of parallel lookups. The left-most points on the graph represent

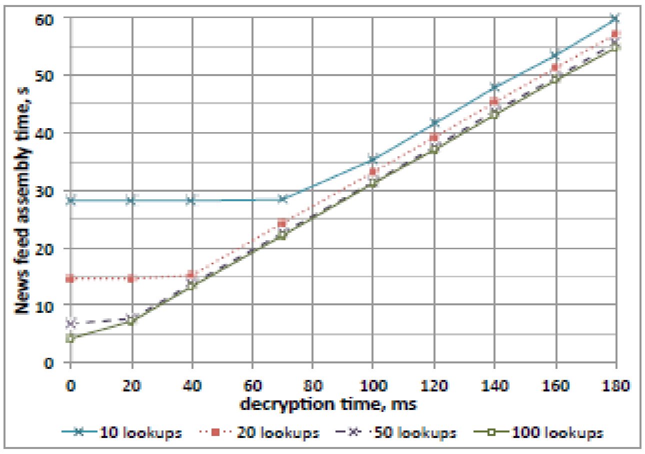

Fig. 2. News feed assembly time for 300 profiles

the case when decryption is instantaneous, but DHT lookups still require some time. Since decryption runs in parallel with DHT lookups, its impact on the total assembly time is barely visible up to some decryption speed after which decryption starts to slow down the newsfeed assembly. Under a condition that one decryption operation takes 70 ms (PE decryption with 10 attributes) a mean news feed assembly time for the case with 100 parallel lookups is 22.1 seconds with a standard deviation of 1.1 seconds. A mean number of decrypted objects in the first two seconds is 25, which we consider acceptable because the following news feed assembly can run in the background and show the results to the user as they come. For comparison, according to [30], the median time to load a Facebook page without images and stylesheets is 3.3 seconds. Facebook also uses a technique of partial content loading [31]. When a user logs in, for example, only 15 news feed stories are loaded. Each time a user scrolls down, a new chunk of user stories is loaded [32]. Thus, while a decryption time of 70 ms may sound large in isolation, we believe that when network latency and UI optimizations are taken into consideration, it still allows for a good user experience.

To reduce the newsfeed assembly time for the particular scheme we need to increase the decryption speed. It can be achieved by performing decryption on multiple processor cores (we were using only one single-threaded decryption engine for this simulation). One could also shift to a push strategy, immediately sending news to friends' accounts when they arise. The news message is encrypted on a per-friendship basis with a corresponding shared secret key. It takes two lookups to get a link to the "incoming" file. To push a news message to all 300 friends (running 100 lookups in parallel) with assumption that symmetric encryption is instantaneous requires a few seconds. The drawback of this strategy is that the amount of required storage space is increased, even if only the url to the news message and not the message itself is sent.

V.CONCLUSION

We have proposed to apply a privacy preserving scheme to the DOSN context: inner-product predicate encryption (PE). It is too expensive to use out of the box. Therefore for PE we proposed a construction for access policies that drastically increases performance, but introduces some trade-offs: it allows encrypting for a bounded set of groups/users; this bound is a trade-off between efficiency and functionality of the scheme; the number of groups in the system is unlimited; a user has 2g different decryption keys, where g is the number of groups a user is a member of having multiple keys leaks some information about access policies. PE is most suitable for encrypting for groups or small sets of separate identities.

We designed an experiment that showed that for newsfeed assembly from all friends our scheme shows good performance and thus user experience.

For schemes that do not reveal access policies and have relatively slow decryption, we proposed to use Bloom filters to indicate to users which files they can decrypt. Bloom filters are both performance and space-efficient, and thus are suitable for DOSNs.

In this paper, we focused the evaluation on performance to see if PE is even feasible under the constraints of decentralized online social networks, starting from the security and privacy properties of the original scheme. The next steps are to focus on security and privacy, as well as semantics of access policies of our modifications.

ACKNOWLEDGMENT

This research has been funded by the Swedish Foundation for Strategic Research grant SSF FFL09-0086 and the Swedish Research Council grant VR 2009-3793.

REFERENCES

[1] G. Greenwald and E. MacAskill, "NSA prism program taps in to user data of apple, google and others," 2013. [Online].Available:http://www.guardian.co.uk/world/2013/jun/06/us-tech-giants-nsa-data

[2] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attributebased encryption," in Proceedings of the 2007 IEEE Symposium on Security and Privacy, ser. SP '07. IEEE Computer Society, 2007, pp. 321–334. [Online]. Available: http://dx.doi.org/10.1109/SP.2007.11

[3] C. Delerablee, P. Paillier, and D. Pointcheval, "Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys," in Pairing-Based Cryptography Pairing 2007, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, vol. 4575, pp. 39–59.

[4] T. Nishide, K. Yoneyama, and K. Ohta, "Attribute-based encryption with partially hidden encryptor-specified access structures," in ACNS, ser. LNCS, vol. 5037. Springer-Verlag, 2008, pp. 111–129.

[5] B. Libert, K. G. Paterson, and E. A. Quaglia, "Anonymous broadcast encryption: adaptive security and efficient constructions in the standard model," in PKC, ser. LNCS, vol. 7293. Springer-Verlag, 2012.

[6] L. Cutillo, R. Molva, and T. Strufe, "Safebook: A privacy-preserving online social network leveraging on real-life trust," Communications Magazine, IEEE, vol. 47, no. 12, pp. 94 –101, dec. 2009.

[7] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: an online social network with user-defined privacy," SIGCOMM Comput. Commun. Rev., vol. 39, pp. 135–146, August 2009.

[8] S. Nilizadeh, S. Jahid, P. Mittal, N. Borisov, and A. Kapadia, "Cachet: a decentralized architecture for privacy preserving social networking with caching," in CoNEXT. ACM, 2012, pp. 337–348.

[9] F. G¨unther, M. Manulis, and T. Strufe, "Cryptographic treatment of private user profiles," in Financial Cryptography, ser. LNCS, vol. 7126. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 40–54.

[10] Y. Afify, "Access control in a peer-to-peer social network," Master's thesis, EPFL, Lausanne, Switzerland, 2008.

[11] S. Buchegger, D. Schi¨oberg, L.-H. Vu, and A. Datta, "Peerson: P2P social networking: early experiences and insights," in Proceedings of the Second ACM EuroSys Workshop on Social Network Systems, ser. SNS '09, 2009, pp. 46–52.

[12] S. Jahid, S. Nilizadeh, P. Mittal, N. Borisov, and A. Kapadia, "Decent: A decentralized architecture for enforcing privacy in online social networks," in PerCom Workshops, 2012, pp. 326–332.

[13] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: The least-authority filesystem," in Proceedings of the 4th ACM International Workshop on Storage Security and Survivability, ser. StorageSS '08. New York, NY, USA: ACM, 2008, pp. 21–26.

[14] J. Anderson, C. Diaz, J. Bonneau, and F. Stajano, "Privacy-enabling social networking over untrusted networks," in Proceedings of the 2Nd ACM Workshop on Online Social Networks, ser. WOSN '09. New York, NY, USA: ACM, 2009, pp. 1–6.

[15] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in EUROCRYPT, ser. LNCS, N. Smart, Ed. Springer-Verlag, 2008, vol. 4965,pp.146–162.

Selected Paper from International Conference on Computing (NECICC-2k15)