IBBE-SGX: Cryptographic Group Access Control using Trusted Execution Environments

Stefan Contiu*‡, Rafael Pires†, Sébastien Vaucher†, Marcelo Pasin†, Pascal Felber† and Laurent Réveillère*

*University of Bordeaux, France, firstname.lastname@u-bordeaux.fr

†University of Neuchâtel, Switzerland, firstname.lastname@unine.ch

‡Scille SAS, France

***Abstract*—While many cloud storage systems allow users to protect their data by making use of encryption, only few support collaborative editing on that data. A major challenge for enabling such collaboration is the need to enforce cryptographic access control policies in a secure and efficient manner. In this paper, we introduce IBBE-SGX, a new cryptographic access control extension that is efficient both in terms of computation and storage even when processing large and dynamic workloads of membership operations, while at the same time offering *zero knowledge* guarantees.**

IBBE-SGX builds upon Identity-Based Broadcasting Encryption (IBBE). We address IBBE's impracticality for cloud deployments by exploiting Intel Software Guard Extensions (SGX) to derive cuts in the computational complexity. Moreover, we propose a group partitioning mechanism such that the computational cost of membership update is bound to a fixed constant partition size rather than the size of the whole group. We have implemented and evaluated our new access control extension. Results highlight that IBBE-SGX performs membership changes 1.2 orders of magnitude faster than the traditional approach of Hybrid Encryption (HE), producing group metadata that are 6 orders of magnitude smaller than HE, while at the same time offering *zero knowledge* guarantees.

I. Introduction

Cloud storage services such as Amazon Web Services, Google Cloud Platform or Microsoft Azure have shown rapid adoption during the last years [1]. However, they lack in offering trustworthiness and confidentiality guarantees to end users. Although threats commonly originate from malicious adversaries breaching security measures to gain access to user data, the menace can also come from an employee with generous privileges, or curious governments warranting data collection for national interest. To overcome these issues, many approaches rely on the construction of cryptographic solutions in which the data is secured on the client side before reaching the storage premises [2], therefore mitigating the lack of trust in the cloud provider.

To enable collaborative operations on the already secured data, one needs to enforce access control policies. Because of the untrusted nature of cloud storage, such administrative operations also need to be cryptographically protected. Enforcing cryptographic access control on an untrusted cloud storage context is subject to a number of requirements. First, access control schemes should incur as low traffic overhead as possible because cloud storages have slower response times in comparison to traditional storage mediums. Second, since a realistic and dynamic membership operations pattern [3] coupled with large volumes of users can make the system unusable in practice, the system must be limited to an acceptable computational bound. Third, as only privileged users perform access control operations, it is required that they gain *zero knowledge* to the data content to which the policy is applied. Last, identities normally employed by users when interacting with the cloud storage (i.e. existing credentials) should be sufficient for membership operations, hence avoiding complex trust establishment protocols.

A number of cryptographic constructions have been proposed for achieving access control. The simplest one, popularly referred to as Hybrid Encryption (HE), makes use of symmetric and public-key cryptography, by employing the former on the actual data and the latter on the symmetric key [4]. Other approaches rely on pairing-based cryptography as a substitute for public-key cryptography, and offer different levels of granularity for specifying access control policies. A few examples include: Identity Based Encryption (IBE) [5] which works similarly to public-key encryption at the identity level; Attribute-Based Encryption (ABE) [6] that supports a richer tree-like access policy expressiveness; or Identity-Based Broadcasting Encryption (IBBE) [7] that can capture group-like policies. Similarly, *Identity-based proxy re-encryption* relies on a semi-trusted middle entity to whom users delegate the re-encryption rights [8].

Unfortunately, pairing-based constructions suffer from important performance issues. According to *Garrison et al.* [3], they are an order of magnitude slower than public-key cryptography. Remarkably, even Hybrid Encryption with Public Key (HE-PKI) incurs prohibitive costs for dynamic access control (see Figure 2). Moreover, the aforementioned constructions do not guarantee our *zero knowledge* requirement.

In this paper, we introduce a new cryptographic access control scheme that is both computationally- and storage-efficient considering a dynamic and large set of membership operations, while offering *zero knowledge* guarantees. *Zero knowledge* is guaranteed by executing the cryptographic access control membership operations in a Trusted Execution Environment (TEE).[1] Our scheme is based on IBBE which is known to be flexible enough to produce small constant policy sizes. Its main drawback is its high impracticable computational cost. Our solution is to execute the membership operations

[1]Only membership operations rely on the TEE; user operations are done in a conventional execution environment.

of IBBE within the TEE so that we can make use of a master secret key. The TEE guarantees that this secret stays within the trusted computing boundary. We can therefore propose an optimization of a well-studied IBBE scheme [9] that drastically reduces its computational complexity. A remaining issue is the computational complexity required for users to derive membership changes. To mitigate this last aspect, we propose a group partitioning mechanism such that the computational cost on the user-side is bound to a fixed constant partition size rather than the potential large group size.

We have implemented our new access control scheme using Intel Software Guard Extensions (SGX) as TEE. To the best of our knowledge, we are the first to adapt the pairing-based-specific library PBC [10] and its underlying dependency GMP [11] to accurately run within SGX. Moreover, we deployed our system on a commercially-available public cloud storage. Our evaluation shows that our scheme only requires few resources while performing better than HE, in addition to providing *zero knowledge*.

To summarize, our contributions are the following:

- We propose a new approach to IBBE by confiding in Intel SGX. To the best of our knowledge, this is the first effort seeking to lower the computational complexity bound of a well-studied IBBE scheme using TEEs as enabling technology. Additionally, our scheme only requires TEE support for a minimal set of users (i.e. administrators).
- We instantiate the novel IBBE-SGX construction to an access control system hosted on a honest-but-curious cloud storage, proposing an original partitioning scheme that lowers the time required by users to ingest access control changes.
- We fully implemented our original access control system and evaluated it against a realistic setup. We conducted extensive evaluations, showing that our system surpasses the performances obtained using state-of-the-art approaches.

Even though the main motivation for this work is to securely share data in a cloud environment, the proposed solution can be applied for encrypting arbitrary information that is securely broadcasted to a group of users over any shared media. Some other examples, besides cloud storages, are peer-to-peer networks or pay-per-view TV.

The rest of the paper is organized as follows. Section II presents the model assumptions undertaken within the problem context. We provide some background about Intel SGX and cryptographic schemes in Section III. Section IV presents the unique constructions that allow us to lower the complexity of an access control scheme by relying on a TEE. In the second part of that section, details about the partitioning mechanism are shown. We describe the design and implementation of an end-to-end system built on top of the scheme in section V. Section VI presents the evaluation of our solution by performing both micro- and macro-benchmarks. Section VII presents the related work in the fields of cryptography, access control systems and SGX. Finally, Section VIII concludes and presents future work.

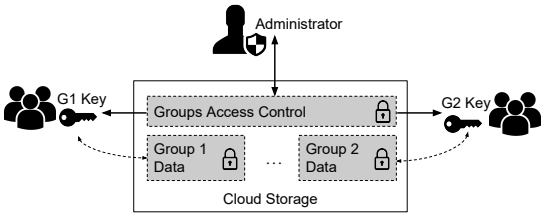


Fig. 1. Model diagram.

II. Model

In this study, we consider groups of users who perform collaborative editing on cryptographically-protected data stored on untrusted cloud storage systems. The data is protected using a block cipher encryption algorithm such as Advanced Encryption Standard (AES) making use of a symmetric group key `gk`. As illustrated in Figure 1, this work addresses the challenge of designing a system for group access control, in which the group key `gk` is cryptographically protected and derivable only by the members of the group. Because groups may become large with a significant turnover in their members, we investigate the implication of numerous member additions and revocations happening throughout their lifetimes.

We distinguish between two types of actors interacting within the system: *administrators* and *users*. All group membership operations are performed by administrators. Their duties include creating groups, and adding or revoking group members. The administrators manifest an honest-but-curious behavior, correctly serving work requests but with a possible malicious intent of discovering the group key `gk`. On the other hand, users listen to the cloud storage for group membership changes, and derive the new group key `gk` whenever it changes. Users are considered of having a trusted behavior.

The role of the cloud storage is to store the definitions of groups access control, also referred to as groups metadata, together with the list of members composing the group, and the actual group data. Besides being a storage medium, we also use the cloud storage as a broadcasting interface for group access control changes. Administrators are communicating with the cloud each time a group membership operation takes place so that users can be notified of the group membership update. We consider the cloud storage to show a similar behavior than administrators (i.e. honest-but-curious). It correctly services assigned tasks albeit with a possible malicious intent of peeking into the groups secrets. Moreover, when manifesting the curious behavior, the cloud storage could collude with any number of curious administrators or revoked users.

Size-wise, we target a solution that can accommodate groups of a very large-scale nature.[2] It is desired that administrators perform membership changes for multiple groups at a time, therefore the number of administrators is relatively small when compared to the number of users.

We choose to ensure authenticity guarantees only with respect to administrator identities, and therefore authenticate membership changes operations. Also, authenticating the

[2]Our evaluation uses 1 million users as the largest group size.

group data created by users is out of scope, the current model being focused on confidentiality guarantees. Therefore, the notion of a reference monitor [3], [12] on the cloud storage is not pertinent within our context. Finally, we do not consider hiding the identities of group members, nor the type of executed membership operations, as they can be inferred by the cloud storage from traffic access patterns. Privacy constructions offering such guarantees [13]–[15] are orthogonal to our work.

III. Background

The building blocks that lead to the creation of our access control extension are Intel SGX, Hybrid Encryption, and Identity Based Broadcast Encryption. This background section presents them together with open challenges.

A. Intel SGX

Intel SGX is an instruction extension available on modern x86 CPUs manufactured by Intel. Similarly to ARM Trustzone [16] or Sanctum [17], SGX aims to shield code execution against attacks from privileged code (e.g. infected operating system) and certain physical attacks. A unit of code protected by SGX is called an *enclave*. Computations done inside the enclave cannot be seen from the outside [18]. SGX seamlessly encrypts memory so that plaintext data is only present inside the CPU package. The assumption is that opening the CPU package is difficult for an attacker, and leaves clear evidence of the breach. Encrypted memory is provided in a processor-reserved memory area called the Enclave Page Cache (EPC), which is limited to 128 MB in the current version of SGX.

Intel provides a way for enclaves to *attest* each other [19]. After the attestation process, enclaves will be sure that each other is running the code that they are meant to execute. The attestation process can be extended to *remote attestation* that allows a piece of software running on a different machine to make sure that a given enclave is running on a genuine Intel SGX-capable CPU. An Intel-provided online service — the Intel Attestation Service (IAS) — is used to check the signature affixed to a *quote* created by the CPU [19]. As part of the attestation process, it is possible to provision the enclave with secrets. They will be securely transmitted to the enclave if and only if the remote attestation process succeeds.

The Trusted Computing Base (TCB) of an SGX enclave is composed of the CPU itself, and the code running within. The assumption is that we trust Intel for securely implementing SGX. Nevertheless, it has been shown that SGX is vulnerable to side-channel attacks [20]. We consider this flaw to be orthogonal to our research, and hence do not consider it in our security evaluation.

B. A Naive Approach to Group Access Control

Suppose that we want to come up with a simple, yet secure, cryptographic scheme to protect a group key `gk`. We can make use of an asymmetrical encryption primitive [21], based on RSA or Elliptic Curve Cryptography (ECC). As each user in the system possesses a public-private key pair, the scheme consists in encrypting `gk` using the public key of each member in the group. A member of the group can then deduce `gk` by decrypting the resulting ciphertext using her private key. This construction is sometimes referred to as Hybrid Encryption (HE) [3], or Trivial Broadcast Encryption Scheme [22].

To achieve the *zero knowledge* requirement, administrators could be asked to run HE within an SGX enclave, thus protecting the discovery of `gk`. However, before discussing the cost of such an integration between HE and SGX, we point out a number of prior weaknesses of HE.

First, the amount of group metadata grows linearly with the number of members in the group, making it impractical in the context of very large groups. Second, when revoking group members, a new key `gk` needs to be created; the entire group metadata also needs to be generated again by encrypting the latest value of `gk`. As the group size increases, the computational cost of the scheme grows linearly. Likewise, the latency incurred for putting, getting and storing the group metadata on the cloud storage will also seriously expand.

Furthermore, when performing group membership operations, the administrators need to entrust the authenticity of the public keys linked to the identity of the members. Public Key Infrastructures (PKIs) [21] can be used to solve this issue. Besides the trust risks that the PKI brings [23], one needs to account for the practical costs of setting up, running and accessing a PKI. To mitigate these risks, one could choose to substitute public-key primitives with identity-based ones. Identity Based Encryption (IBE) [5], [24] makes use of arbitrary strings as public keys; we can therefore use a user name directly as a public key. The user secret key is generated at setup phase or later by a Trusted Authority (TA). Obviously, both HE-PKI and Hybrid Encryption with Identity-based Encryption (HE-IBE) have the same inner functioning, when making abstraction of the key methodology choice.

Integrating SGX with HE-PKI and HE-IBE is required in order to guarantee the *zero knowledge* property against administrators. As hybrid encryption is causing a high group metadata expansion, it has a direct impact on the memory space that is used inside the SGX enclave. Accessing memory in SGX enclaves can induce an overhead of up to 19.5 % for write accesses and up to 102 % for read accesses [25]. Apprehensive about the hypothesized SGX degradation in performance caused by the group metadata expansion, we shift the focus on finding a solution with minimal expansion.

C. Broadcast Encryption (BE) and Identity-Based BE

In order to optimize both SGX and cloud transit costs, we investigate the possibility of cryptographic schemes that induce a minimal group expansion.

Broadcast Encryption (BE) [26] is a public-key cryptosystem with a unique public key that envelopes the entire system, contrary to the HE scheme where each user uses a different public key. However, each user in a BE system has a unique private key generated by a trusted authority. To randomly generate a group key `gk` and the associated group metadata (named *encrypt* operation within BE systems), one makes use

of the system-wide public key. On the other side, when a user wants to reveal `gk` (*decrypt* in BE systems), she makes use of her individual private key.

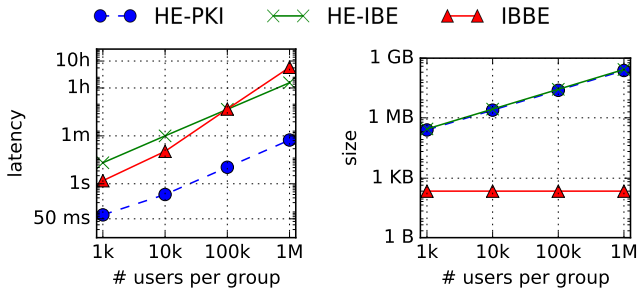
As broadcast encryption schemes come with different contextual models, we impose a number of conditions. First, to maintain the same threat model as HE, we are only investigating the use of fully collusion-resistant BE schemes [27], in which no coalition of members outside of the group could reveal `gk`. Second, the set of users participating in the system is not initially known, thus we rely on the usage of *dynamic* BE schemes [28]. Third, as in the case of HE, we would prefer constructions that can accommodate the use of IBE.

Piercing through the existing research literature, we identified an IBBE scheme [29] that not only fulfills all the aforementioned requirements, but also operates with group metadata expansions and user private keys of constant sizes. Moreover, the scheme has an additional strategic advantage that proves beneficial in our context: the system-wide public key size is linear in the maximal size of any group.

Upon analyzing the computational complexity of the selected IBBE scheme, one can notice that creating `gk` given a set of members, as well as decrypting it as a user, are operations with a quadratic complexity in the number of members. Therefore, even though the scheme brings a tremendous gain in the size of group metadata expansion, the computational cost of IBBE might be excessive for practical use.

Figure 2 exemplifies the performance of HE-PKI, HE-IBE and IBBE schemes in their raw form, before any integration with SGX is considered. The sub-figure on the left displays the total time taken for the operation of creating a group while the one on the right shows the size occupied by the expansion of group metadata. The optimality of IBBE regarding the size of group metadata expansion is immediately obvious. It always produces 256 bytes of metadata, regardless of the number of users per group. That is preferable compared to HE-PKI and HE-IBE, which produce increasingly larger values, as much as 27 MB for groups of 100,000 users, and 274 MB for the largest benchmarked group size. On the other hand, IBBE performs much worse than HE-PKI when considering the execution time. It is 150× and 144× slower for groups of 10,000 and 100,000 users, respectively.

There is no doubt that running the IBBE scheme in this form is inadequate. In the remainder of this paper, we describe two original contributions, one that changes a traditional assumption of the IBBE scheme, and a second that lowers the user decryption time.



(a) Latency for group creation. (b) Group metadata expansion.

Fig. 2. Performance of HE-PKI, HE-IBE and IBBE, without *zero knowledge*.

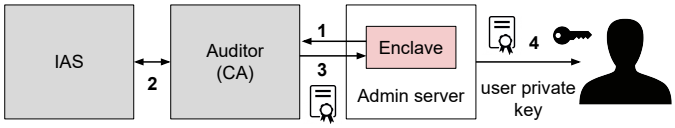


Fig. 3. Initial setup.

IV. IBBE-SGX

IBBE-SGX can be broadly described in 3 steps: (i) trust establishment and private key provisioning; (ii) membership definitions and group key provisioning; and (iii) membership changes and key updates.

A. Trust Establishment

IBBE schemes generate a single public key that can be paired with several private keys, one per user. Users, in turn, need to be sure that the private key they receive is indeed generated by someone they trust, otherwise they would be vulnerable to malicious entities trying to impersonate the key issuer. To achieve that, we rely upon a PKI to provide verifiable private keys to users.

Another security requirement of IBBE-SGX is that the key management must be kept in a TEE. Therefore, there must be a way of checking whether that is the case. On that front, Intel SGX makes it possible to attest enclaves. Running this procedure gives the assurance that a given piece of binary code is truly the one running within an enclave, on a genuine Intel SGX-capable processor (Section III-A).

Figure 3 illustrates the initial setup of trust that must be executed at least once before any key leaves the enclave. Initially, the enclaved code generates a pair of asymmetric keys. While the private one never leaves the trusted domain, the public key is sent along with the enclave measurement to the Auditor (1), who is both responsible for attesting the enclave and signing its certificate, thus also acting as a Certificate Authority (CA). Next, the Auditor checks with IAS (2) if the enclave is genuine. Being the case, it compares the enclave measurement with the expected one, so that it can be sure that the code inside the shielded execution context is trustworthy. Once that is achieved, the CA issues the enclave's certificate (3), which also contains its public key. Finally, users are able to receive their private keys and the enclave's certificate (4). The key will be encrypted by the enclave's private key generated in the beginning. To be sure they are not communicating with rogue key issuers, users check the signature in the certificate and then use the enclave's public key contained within. All communication channels described in this scheme must be encrypted by cryptographic protocols such as Transport Layer Security (TLS).

B. From IBBE to IBBE-SGX

Traditionally, the IBBE scheme [7], [29] consists of the following four operations.

1) System Setup: The system setup operation is run once by a Trusted Authority (TA) and generates a Master Secret Key M_{SK} and a system-wide Public Key PK.

2) Extract User Secret: The TA then uses the Master Secret Key M_{SK} to extract the secret key U_{SK} for each user U.

3) Encrypt Broadcast Key: The broadcaster generates a randomized Broadcast Key b_k for a given set of receivers $\mathcal{S}$, by making use of PK. Together with b_k, the operation outputs a public broadcast ciphertext c. The broadcast ciphertext can be publicly sent to members of $\mathcal{S}$ so they can derive b_k.

4) Decrypt Broadcast Key: Any member of $\mathcal{S}$ can discover b_k by performing the decrypt broadcast key operation given her secret key U_{SK} and $(\mathcal{S}, c)$.

In contrast to traditional IBBE that requires the use of a TA to perform the *System Setup* and *Extract User Secret* operations, we rely on SGX enclaves. Therefore, the master secret key `MSK` used by the two aforementioned operations can be made available in plaintext exclusively inside the enclave, and securely sealed if stored outside of the enclave for persistence reasons.

Similarly to IBBE, the *Encrypt Broadcast Key* and *Decrypt Broadcast Key* operations rely on the system public key PK, and are thus usable by any user of the system.

As opposed to the traditional IBBE usage scenario, our model requires that all group membership changes—generating the group key and metadata—are performed by an *administrator*. Administrators can use the master secret key `MSK` to encrypt, set up the system and extract user keys. The decryption operation, however, remains identical to the traditional IBBE approach, being executed by any arbitrary user. In the remainder of this paper, we refer to our new IBBE scheme as **IBBE-SGX**.

We now describe the computational simplification opportunities introduced by IBBE-SGX compared to IBBE [29]. First, by making use of `MSK` inside the enclave, the complexity of the encryption operation drops from $O(|S|^2)$ for IBBE to $O(|S|)$ for IBBE-SGX, where $|S|$ is the number of users in the broadcast group set. The reason behind the complexity drop is bypassing a polynomial expansion of quadratic cost, necessary in the traditional IBBE assumptions. The reader is directed to Section A-C for the concrete mathematical inference process. We argue that this complexity cut is sufficient to tackle the impracticality of the IBBE scheme emphasized earlier in Figure 2. Second, by relying on `MSK`, one can build efficient access control specific operations, such as adding or removing a user from a broadcast group. IBBE-SGX can accommodate $O(1)$ complexities for both operations, as illustrated in Sections A-E and A-F.

Unfortunately, IBBE-SGX maintains an $O(|S|^2)$ complexity for the user decrypt operation, during which, similarly to IBBE encryption, the algorithm performs a polynomial expansion of quadratic cost. We address this drawback by introducing a partitioning mechanism as described later in Section IV-C.

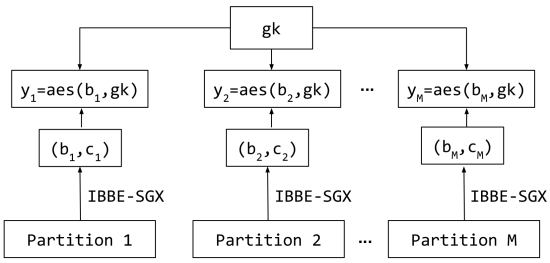


Fig. 4. Partitioning mechanism using IBBE-SGX and AES to protect the Group Key.

Finally, we consider a re-keying operation, for optimally generating a new broadcast key and metadata when the identities of users in the group $\mathcal{S}$ do not change. The operation can be performed in $O(1)$ complexity for both IBBE and IBBE-SGX, as detailed in Section A-G.

C. Partitioning Mechanism for IBBE-SGX

Although IBBE-SGX produces a minimal metadata expansion and offers an optimal cost for group membership operations, it suffers from a prohibitive cost when a member needs to decrypt the broadcast key. To address this issue, we introduce a partitioning mechanism.

As the decryption time is bound to the number of users in the receiving set, we split the group into partitions (sub-groups) and therefore limit the user decryption time to the number of members in a single partition. Moreover, each partition broadcast key will wrap the prime group key `gk`, so that members of different partitions can communicate by making use of `gk`.

The partition mechanism is depicted in Fig 4. The first step consists in splitting the group of users in fixed-size partitions. The administrator can then use the encrypt functionality of IBBE-SGX to generate a sub-group broadcast key b_k and ciphertext c_k for each partition k. Next, for each partition, the group key `gk` is encrypted using symmetric encryption such as AES, by using the partition broadcast key b_k as the symmetric encryption key. Note that since the scheme is executing inside an SGX enclave, a curious administrator cannot observe `gk` nor the broadcast keys.

The group metadata of IBBE-SGX is therefore represented by the set of all pairs composed of the partition ciphertext and the encrypted group key (*i.e.* (c_i, y_i) in Figure 4). The inquisitive cloud storage can then publicly receive and store this set of group metadata.

Whenever a membership change happens, the administrator will update the list of group members and send the affected partition metadata to the cloud. The clients, in turn, can detect a change in their group by listening to updates in their partition metadata.

The partitioning mechanism has an impact on the computational complexity of the IBBE-SGX scheme on the administrator side. First, as the public key PK of the IBBE system is linear in the maximal number of users in a group [29], results that the public key for the IBBE-SGX scheme is linear in the maximal number of users in a partition (denoted

TABLE I
IBBE-SGX AND IBBE OPERATIONS COMPLEXITIES PER THE NUMBER OF PARTITIONS OF A GROUP ($|P|$), THE FIX SIZE OF A PARTITION ($|p|$) AND THE CARDINALITY OF THE GROUP MEMBERS SET ($|S|$).

Operation	IBBE-SGX	IBBE [29]
System Setup	$O(\|p\|)$	$O(\|S\|)$
Extract User Key	$O(1)$	$O(1)$
Create Group Key	$\|P\| \times O(\|p\|)$	$O(\|S\|^2)$
Add User to Group	$O(1)$	
Remove User from Group	$\|P\| \times O(1)$	
Decrypt Group Key	$O(\|p\|^2)$	$O(\|S\|^2)$

by $|p|$). Therefore, both the computational complexity and storage footprint of the system setup phase can be reduced by a factor representing the maximal number of partitions, without losing any security guarantee. Second, the complexities of IBBE-SGX operations change to accommodate the partitioning mechanism, as shown in Table I. Creating a group becomes the cost of creating as many IBBE-SGX partitions that the fixed partition size dictates. Adding a user to a group remains constant, as the new user can be added either to an existing partition or to a brand new one. Removing a user implies performing a constant time re-keying for each partition. Finally, the decryption operation gains by being quadratic in the number of users of the partition rather than the whole group.

The partitioning mechanism also has an impact on the storage footprint for group metadata. Compared to IBBE when considering a single partition, the footprint is augmented by the symmetrically encrypted partition broadcast key (i.e. y_i) and the *nonce* required for this symmetric encryption. When considering an entire group, the cost of storing the group metadata is represented by the cost of a single partition multiplied by the number of partitions in the group, in addition to a metadata structure that keeps the mapping between users and partitions.

Although the partition mechanism induces a slight overhead, the number of partitions in a group is relatively small compared to the group size. Second, partition metadata are only manipulated by administrators, so they can locally cache it and thus bypass the cost of accessing the cloud for metadata structures. Third, as our model accepts that the identities of group members can be discovered by the curious administrator or the cloud, there is no cryptographic operation needed to protect the mappings within the partition metadata structure.

Determining the optimal value for the partition size mainly depends on the dynamics of the group. Indeed, there is a trade-off between the number and frequency of operations performed by the administrator for group membership and those performed by regular users for decrypting the broadcast key. A small partition size reduces the decryption time on the user side while a larger partition size reduces the number of operations performed by the administrator to run IBBE-SGX and to maintain the metadata.

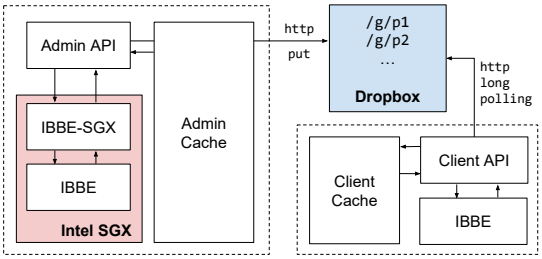


Fig. 5. Big Picture Architecture

V. IBBE-SGX GROUP ACCESS CONTROL SYSTEM

We describe in this section the design and implementation of an end-to-end group access control system based on IBBE-SGX. The overall architecture is illustrated in Figure 5 and consists of a client and an administrator using Dropbox as a public cloud storage provider.

A. System Design

The administrator's Application Programming Interface (API) makes calls to the underlying SGX enclave that hold the functionalities of IBBE-SGX which is built on top of an IBBE component. Since SGX is not required on the client side, the *Client API* directly calls the functionalities of the IBBE component. Both administrators and clients make use of local in-memory caches in order to save round-trips to the cloud for accessing existing access policies. Administrators make use of the `PUT` HTTP verb to send data to the cloud, while clients are listening by using *HTTP long polling*. In Dropbox, *long polling* works at the directory level, so we index the group metadata as a bi-level hierarchy. The parent folder represents the group, and each child stands for a partition.

The operation for creating a group is described in Algorithm 1. Once the fixed-size partitions are determined (line 1), the execution enters the SGX enclave (lines 2 to 6) during which the random group key is enveloped by the hash of each partition broadcast key. The ciphertext values, as well as the sealed group key, leave the enclave to be later pushed to the cache and the cloud (line 7).

Algorithm 1 Create Group

Input: Group g, Members $S = \{u_1, ..., u_n\}$, Partition size m

1: $\mathcal{P} \leftarrow \{\{u_1, ..., u_m\}, \{u_{m+1}, ..., u_{2m}\}, ...\}$

Enclaved

2: $gk \leftarrow RandomKey()$
3: **for** $p \in \mathcal{P}$ **do**
4: $\quad (b_p, c_p) \leftarrow sgx\_ibbe\_create\_partition(\texttt{MSK}, p)$
5: $\quad y_p \leftarrow sgx\_aes(sgx\_sha(b_p), gk)$
6: $sealed\_gk \leftarrow sgx\_seal(gk)$

7: Store: (1) $sealed\_gk$; (2) $\forall p \in P : \langle \forall u \in p, y_p, c_p \rangle$

The operation of adding a user to a group (Algorithm 2) starts by finding the set of all partitions with remaining capacity (line 1). If no such a partition is found, a new partition is created for the user (line 3) and the group key is enveloped

by the broadcast key of the new partition (lines 4 to 6), before persisting its ciphertexts (line 7). Otherwise, a partition that is not empty is randomly picked, and the user is added to it (lines 9, 10). Since the partition broadcast key remains unchanged, only the ciphertext needs to be adapted to include the new user (line 10). The partition members and ciphertext are then updated on the cloud (line 12). Note that there is no need to push the encrypted group key y_{add} as it was not changed.

Algorithm 2 Add User to Group

Input: Group: g, Partitions of g: $\mathcal{P}$, User to add: u_{add}, Sealed group key: $sealed\_gk$.

1: $\mathcal{P}' \leftarrow \forall p \in \mathcal{P},\ such\ that\ |p| < m.$
2: **if** $\mathcal{P}' = \emptyset$ **then**
3: $\quad p_{add} \leftarrow \{u_{add}\}$

Enclaved

4: $\quad (b_{add}, c_{add}) \leftarrow sgx\_ibbe\_create\_partition(\texttt{MSK}, p_{add})$
5: $\quad gk \leftarrow sgx\_unseal(sealed\_gk)$
6: $\quad y_{add} \leftarrow sgx\_aes(sgx\_sha(b_{add}), gk)$

7: $\quad$ Store: $\langle \{u_{add}\}, y_{add}, c_{add} \rangle$
8: **else**
9: $\quad p_{add} \leftarrow RandomItem(\mathcal{P}')$
10: $\quad p_{add} \leftarrow p_u \cup \{u_{add}\}$

11: $\quad c_{add} \leftarrow sgx\_add\_user\_to\_partition(\texttt{MSK}, p_{add}, u_{add})$

12: $\quad$ Update: $\langle \forall u \in p_{add}, *, c_{add} \rangle$
13: $\mathcal{P} \leftarrow p_{add} \cup \mathcal{P}$

Removing a user from a group (Algorithm 3) proceeds by removing the user from her hosting partition (lines 1 and 2). Next, a new group key is randomly generated (line 3). The former user hosting partition broadcast key and ciphertext are changed to reflect the user removal (line 4) and then used for enveloping the new group key (line 5). For all the remaining partitions, a constant time re-keying regenerates the partition broadcast key and ciphertext that envelopes the new group key (lines 6 to 9). After sealing the new group key (line 10), the changes of metadata for the group partitions are pushed to the cloud (line 11). Note that the partition members only need to be updated for the removed user hosting partition.

Algorithm 3 Remove User u_{rem} from Group g

Input: Group: g, Partitions of g: $\mathcal{P}$, User to remove: u_{rem}.

1: $p_{rem} \leftarrow p \in \mathcal{P},\ such\ that\ u_{rem} \in p.$
2: $p_{rem} \leftarrow p_{rem} \setminus \{u_{rem}\}$

Enclaved

3: $gk \leftarrow RandomKey()$
4: $(b_{rem}, c_{rem}) \leftarrow sgx\_remove\_user(\texttt{MSK}, p_{rem}, u_{rem})$
5: $y_{rem} \leftarrow sgx\_aes(sgx\_sha(b_{rem}), gk)$
6: **for** $p \in \mathcal{P} \setminus p_{rem}$ **do**
7: $\quad (b_p, c_p) \leftarrow sgx\_rekey\_partition(p)$
8: $\quad y_p \leftarrow sgx\_aes(sgx\_sha(b_p), gk)$
9: $sealed\_gk \leftarrow sgx\_seal(gk)$

10: Update: (1) $\langle \forall u_i \in p_{rem}, y_{rem}, c_{rem} \rangle$
11: $\quad$ (2) $\forall p \in \mathcal{P} \setminus p_{rem} : \langle *, y_p, c_p \rangle$

As many removal operations can result in partially unoccupied partitions, we propose the use of a re-partitioning scheme whenever the partition occupancies are too low. We implement a heuristic to detect a low occupancy factor such that if less than half of the partitions are only two thirds full, then re-partitioning is triggered. Re-partitioning consists in simply re-creating the group following Algorithm 1.

Finally, the client decrypt operation works by first using IBBE to decrypt the broadcast key and then use the hash of this key for an AES decryption to obtain the group key. Due to space constraints, we omit the formal algorithm specification.

B. Implementation

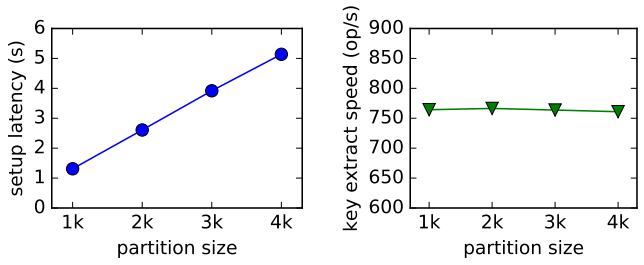
In order to implement the system, we used the PBC [10] *pairing-based cryptography* library which, in turn, depends on GMP [11] to perform arbitrary precision arithmetics. They both have to be used inside SGX enclaves (Section IV). There are several challenges when porting legacy code to run inside enclaves. Besides having severe memory limitations (Section III-A), it also considers privileged code running in any protection ring but user-mode (ring 3) as not trusted. Therefore, enclaves cannot call operating system routines.

Although memory limitations can have performance implications at runtime, they have little influence on enclave code porting. Calls to the operating system, on the other hand, can render this task very complex or even unfeasible. Luckily, since both PBC and GMP mostly perform computations rather than input and output operations, the challenges on adapting them were chiefly restrained to tracking and adapting calls to *glibc*. The adaptations needed were done either by relaying operations to the operating system through *outside calls* (ocalls), or performing them with enclaved equivalents. The outside calls, however, do not perform any sensitive action that could compromise security. Aside from source code modifications, we dedicated efforts to adapt the compilation toolchain. This happens because one has to use curated versions of standard libraries (like the ones provided by Intel SGX SDK), besides having to prevent the use of compiler's built-in functions and setting some other code generation flags. The total number of Lines of Code (LoCs) or compilation toolchain files that were modified were 32 lines for PBC and 299 for GMP.

Apart from changes imposed by SGX, we also needed to use common cryptographic libraries. Although some functions are provided in v.1.9 of the Intel SGX SDK [19], its AES implementation is limited to 128 bits. Since we aim at the maximal security level, we used the AES 256 bits implementation provided in Intel's port of OpenSSL [30]. The end-to-end system encapsulating both IBBE-SGX and HE schemes consists in 3,152 lines of C/C++ code and 170 lines of Python.

VI. Evaluation

In this section, we benchmark the performance of the IBBE-SGX scheme from three different perspectives: by measuring the operations performance in isolation, then by comparing them to Hybrid Encryption (HE), and finally by capturing the performance when replaying realistic and generated access control traces. We chose to compare IBBE-SGX to HE only



(a) System Set-Up Latency (b) Key Extract Throughput

Fig. 6. Performance of the system bootstrap phase.

as the latter already shows better computational complexity than IBBE (see Figure 2a).

The experiments were performed on a quad-core Intel i7-6600U machine, having a processor at 3.4 GHz with 16 GB of RAM, using Ubuntu 16.04 LTS.

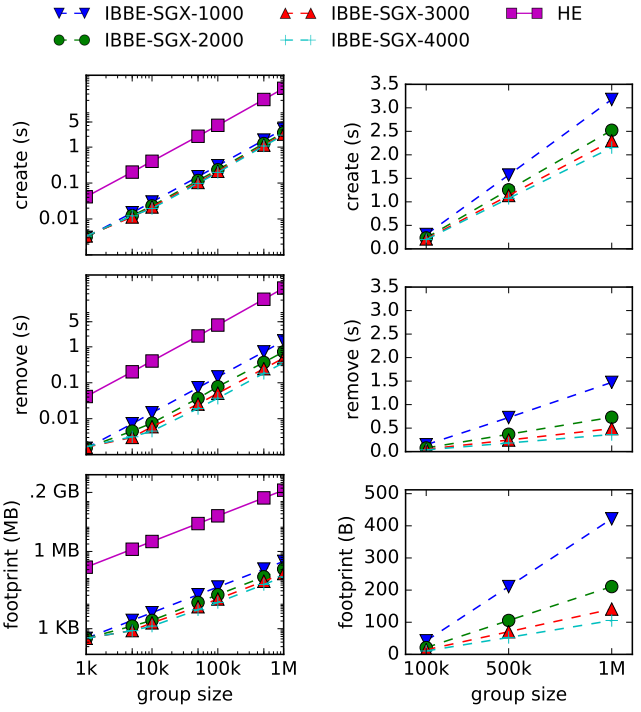
A. Microbenchmarks

Within the microbenchmarks we isolate the performance of each IBBE-SGX operation, and perform a comparison with the HE scheme.

First, we evaluate the performance of the bootstrap phase. It consists on setting up the system and generating secret user keys, referenced in Figure 6. One can notice that the setup phase latency increases linearly per partition size, with a growth of 1.2s per 1,000 users. In contrast, extracting secret user keys gives an average throughput of 764 operations per second, independent of the partition size.

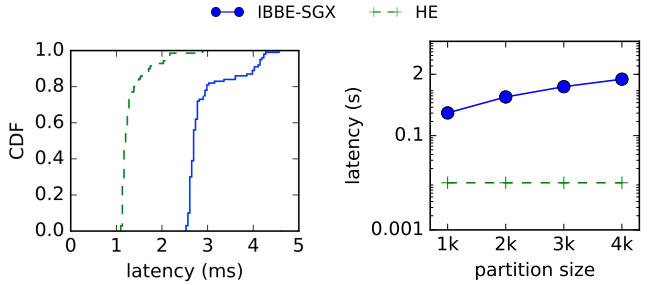
Next, we evaluate the behavior of IBBE-SGX operations compared to HE. Figure 7a displays the computational cost for operations of creating a group, removing a user from a group, and the storage footprint of the group metadata. One can notice that all three operations are better than their HE counterparts by approximately a constant factor. The computational cost of create and remove operations of IBBE-SGX is on average 1.2 orders of magnitude faster than HE. Compared to the original IBBE scheme, IBBE-SGX is better by 2.4 orders of magnitude for groups of 1,000 users and 3.9 orders of magnitude for one million users (see Figures 2a and 7a). Storage-wise, IBBE-SGX is up to 6 orders of magnitude better than HE. Moreover, Figure 7b zooms into the performances of IBBE-SGX create and remove operations, and the storage footprint respectively, when considering different sizes of partitions. One can notice that the remove operation takes half the time than the create operation. Considering the storage footprint, the degradation brought by using smaller partition sizes is fairly small (e.g., 432 vs. 128 bytes for groups of 1 million members).

The Cumulative Density Function (CDF) of latencies for adding a user to a group is shown in Figure 8a. The operation has a constant time complexity for both IBBE-SGX and HE. As the add operation of IBBE-SGX can take two paths, either adding a user to an existing partition or creating a new one if all the others are full, the plot points the difference between the two at the CDF value of 0.8. Moreover, the HE add operation is generally twice as fast as IBBE-SGX.



(a) Comparing IBBE-SGX and HE (b) Partitioning Evaluation

Fig. 7. Evaluation of create and remove operations and storage footprint, by varying the partition size for IBBE-SGX (1000, 2000, 3000 and 4000).



(a) Add user to group performance. (b) Decrypt performance.

Fig. 8. Performance of the adding a user to a group and decrypt operations.

The client decryption performance is shown in Figure 8b. The decryption operation, like the add operation, is faster within the HE approach than IBBE-SGX. The difference of 2 orders of magnitude is caused by the quadratic cost of the IBBE-SGX decryption operation. We argue that a slower decryption time for IBBE-SGX can be acceptable in practice. First, the decrypt performance is overshadowed by the slow cloud response time necessary for clients to update the group metadata that always precedes a decryption operation. Second, the cost of decryption remains bounded to a partition size, independent on the number of users in the group.

B. Macrobenchmarks

1) Real Data Set: To capture the performance of the IBBE-SGX scheme within a realistic scenario, we decide to replay an access control trace based on the membership changes in the version control repository of the Linux Kernel [31].

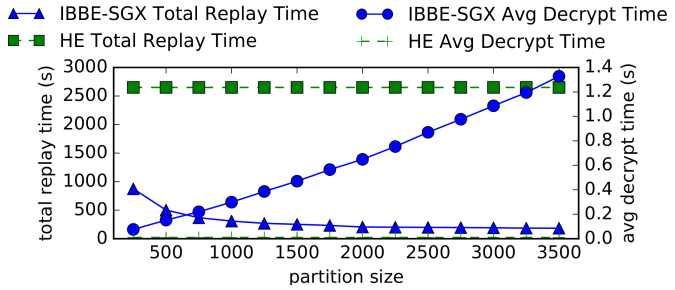


Fig. 9. Measuring total administrator replay time and average user decryption time per different partition sizes using the Linux Kernel ACL data set.

We derive the membership trace by considering the first commit of a user as the add to group operation. The remove from group operation is represented by the user's last commit. The generated trace contains 43,468 membership operations that spawn across a period of 10 years, during which the group size never exceeds 2803 users. We replay the generated trace sequentially for both HE and IBBE-SGX by varying the partition size. We also capture the total time spent by the administrator to replay the trace and the average user decryption time.

Figure 9 displays the results. Considering the administrator replay time, IBBE-SGX performs better when the partition size converges to the number of users in the group. Using a small partition size, e.g. 250, is almost twice as inefficient when compared to a partition of 1000 users. Compared to HE replay time, IBBE-SGX is generally 1 order of magnitude faster. On the other hand, decryption time for IBBE-SGX grows quadratically per partition size while in HE it remains constant. This evidentiates IBBE-SGX's trade-off caused by different partition sizes on the performances of membership changes and user decryption time. A prior estimation of the maximal group size (2803 in our case) suggests the choice of a small partition for practical use (such as 750), so that it can manifest satisfactory outcomes both in terms of admininistrator performance and user decryption time.

2) Synthetic Data Set: In order to observe the impact of different workloads of group membership access control, we generate a set of synthetic traces that capture incremental percentages of revocation rates. Concretely, we generate 11 different traces consisting of 10,000 membership operations. The composition of the traces is randomly generated by considering different revocation rates. We replay the 11 traces through our system and measure the end-to-end time required by the administrator to perform all membership changes. We then repeat the process by considering different partition sizes.

The results are shown in Figure 10. We observe a linear increase in the total time when incrementally increasing the revocation ratio up to 50% in workloads dominated by add operations. After this point, the total time stabilizes and finally decreases when the revocation ratio is more than 90%. This behavior is caused by the merging of sparse partitions, which happens more frequently with the increasing rate of revocations. Having fewer partitions, IBBE-SGX's operations become faster, therefore decreasing the total time.

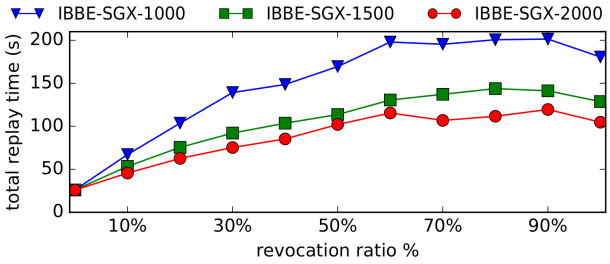


Fig. 10. Measuring total replay time of IBBE-SGX scheme per different partition sizes (1000, 1500, 2000) for synthetically generated datasets considering increasing revocation rates.

VII. Related Work

We structure the presentation of the related work on three axes detailing first research work on cryptographic schemes used for access control. Then we go into research work of systems that cryptographically protect from untrusted storages. Last, we detail related work regarding Intel SGX.

A. Cryptographic Schemes for Access Control

HE making use of a PKI and IBE has been utilized within a role based access control and proved unsuitable in the cloud storage context [3].

ABE [32] is a cryptographic construction that allows a fine-grained access control by matching attributes labeled to both users and content. Depending on the labeled location, one can distinguish between key-policy ABE [6] and ciphertext-policy ABE [33]. However, when employed for simple access control policies, such as our group sharing context, ABE has substantially greater costs than identity-based encryption [3].

Hierarchical Identity Based Encryption (HIBE) [34] and Functional Encryption (FE) [35] are two cryptographic schemes offering functionalities for access control that, similarly to IBE and ABE, rely on pairing-based cryptography. HIBE is specifically designed to target hierarchical organizations where a notion of descendants exists. FE is a powerful construction that can arbitrarily encapsulate programs as access control, but is unsuitable for practical use [36].

Proxy re-encryption [37] is a cryptographic system in which the owner of some encrypted data can delegate the re-encryption of her data to a proxy, with the intent of sharing it with another user. For the re-encryption to take place, the data owner needs to generate and transmit to the proxy a re-encryption key. The scheme proves to be beneficial for the cloud environment, as the re-encryption and the storage of the data can happen on the same premises. A number of approaches have shown how proxy re-encryption can be combined with identity-based encryption [8], or with attribute-based encryption [38], [39]. Differently than proxy re-encryption, our construction does not require users to send transformational keys to the administrators. Therefore, even if the administrators would be hosted on the cloud storage premises, they do not act as proxies.

The related research area of ***multicast communication security*** [22], [40] defines efficient schemes focusing exclusively

on revocation aspects. Logical Key Hierarchy [41] is a rekeying scheme in which communications for revocation operations are minimized to logarithmic sizes. Other schemes [26], [42] exploit a secret sharing mechanism, considering that no coalition of revoked users larger than a threshold number is trying to decrypt the transmission.

B. Cryptographically Protected Untrusted Storages

The shared cloud-backed file system (SCFS) designed by *Bessani et al.* [2] offers confidentiality guarantees to users by encrypting data stored by the clouds on the client-side. Even though the encryption keys are distributed among multiple cloud storages through secret sharing schemes, the access control is not cryptographically protected, but stored and enforced from a trusted coordination service. We argue that this approach is not secure enough because it does not protect from curious administrators. The global access control structure can be compromised if an attacker gain access to this service.

CloudProof [43] is a secure cloud storage system offering guarantees such as confidentiality, integrity, freshness and write-serializability. To enforce access control, CloudProof makes use of broadcast encryption to protect the keys that are used for encrypting and signing the actual data. Unlike our construction, CloudProof does not offer the *zero knowledge* guarantee for membership operations. Moreover, CloudProof does not discuss how the authentic identity of the users in the broadcast set is established (*e.g.* during a group creation operation). Hypothetically, a PKI could be employed for this task, thus requiring a trusted entity in the system. However accessing regularly the PKI would add a significant overhead. In order to mitigate these issues, our solution relies on the identity-based version of broadcast encryption.

REED [44] considers the problem of rekeying in the context of honest-but-curious deduplicated storages. To provide access control, REED relies on ABE. However, as noted by the authors in their evaluation, the performance overhead of the rekeying operation drastically increase to several seconds when varying the total number of users up to 500. Considering group sizes of thousands of users (as we do for groups up to one million), ABE becomes impracticable for access control at large scale.

Sieve [45] platform allows users to store their data encrypted in the cloud and then discretionary delegate access to the data to consuming web services. Sieve makes use of attribute based encryption for access control policies and key homomorphism for providing a *zero knowledge* guarantee against the storage provider. This access control construction has many similarities with ours, however we differentiate exploiting the *zero knowledge* guarantee for lowering the computational complexity of the access control scheme, IBBE in our case.

C. SGX

SGX has been extensively used in shielding applications and infrastructure platform services like ours that handle sensitive data. Iron [36] is the closest to our proposal in the sense that it takes advantage of SGX to build a practical encryption scheme for an unpractical strategy thus far. Like us, they use an enclave that holds a master secret as root for later key derivations. They target, however, functional encryption. The enclave generates a key that is associated to a function, so that the computation can be performed without revealing the data on top of which it is applied. The results of applying such function, though, are presented in clear. The authors show that this approach outperforms by orders of magnitude other cryptographic schemes that also offer functional encryption.

Other systems relate to ours with regards to the reduction of overhead for an otherwise costlier design. Hybster [46], for instance, proposes a hybrid state-machine replication protocol. Hybrid because it does tolerate arbitrary faults but yet it assumes that some nodes may crash. It relies on SGX features such as isolation, replay protection and trusted counters to achieve a parallelization scheme that makes it a viable solution for demanding applications, reaching higher numbers of operations per second in comparison to traditional approaches.

At the level of infrastructure services, SCBR [47] proposes a content-based routing solution where the filtering step is put inside enclaves, thus allowing the matching of publications against stored subscriptions in a safe manner. It is shown to be one order of magnitude faster than an approach with comparable security guarantees. The gain comes from the plaintext operations done inside the enclave against the counterpart that needs to perform computations over encrypted data.

VIII. Conclusion

We have introduced IBBE-SGX, a new cryptographic access control extension that is built upon Identity-Based Broadcasting Encryption (IBBE) and exploits Intel Software Guard Extensions (SGX) to derive cuts in the computational complexity of IBBE. We propose a group partitioning mechanism such that the computational cost of membership update is bound to a fixed constant partition size rather than the size of the whole group. We have implemented and evaluated our new access control extension in a single administrator with multiple users set-up. We have conducted both real and synthetic benchmarks, demonstrating that IBBE-SGX is efficient both in terms of computation and storage even when processing large and dynamic workloads of membership operations. Our innovative construction performs membership changes 1.2 orders of magnitude faster than the traditional approach of Hybrid Encryption (HE), producing group metadata that are 6 orders of magnitude smaller than HE, while at the same time offering *zero knowledge* guarantees.

There are a number of interesting avenues of future work. The first is to dynamically adapt the partition sizes based on the undergoing workload. This would optimize the speed of administrator- and user-performed operations. A second challenge would be to adapt our construction to a distributed set of administrators that would perform membership changes concurrently on the same group or partition, by using lock-free techniques. Third, in a setup with multiple administrators, one can envision certifying blocks of membership operations logs through blockchain-like technologies.

ACKNOWLEDGMENT

The research leading to these results has received funding from the French Directorate General of Armaments (DGA) under contract RAPID-172906010. The work was also supported by European Commission, Information and Communication Technologies, H2020-ICT-2015 under grant agreement number 690111 (SecureCloud project) and partially supported by the CHIST-ERA "DIONASYS" project.

REFERENCES

[1] H. Seybert and P. Reinecke, "Internet and cloud services – statistics on the use by individuals," vol. 16, 2014.

[2] A. N. Bessani, R. Mendes, T. Oliveira, N. F. Neves, M. Correia, M. Pasin, and P. Verissimo, "SCFS: A shared cloud-backed file system.," in *USENIX Annual Technical Conference*, 2014, pp. 169–180.

[3] W. C. Garrison, A. Shull, S. Myers, and A. J. Lee, "On the practicality of cryptographically enforcing dynamic access control policies in the cloud," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 819–838. DOI: 10.1109/SP.2016.54.

[4] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh, "SiRiUS: Securing remote untrusted storage," in *NDSS*, vol. 3, 2003, pp. 131–145.

[5] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," in *CRYPTO 2001*, Springer, 2001, pp. 213–229.

[6] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *CCS'06*, ACM, 2006, pp. 89–98.

[7] R. Sakai and J. Furukawa, "Identity-based broadcast encryption," *IACR Cryptology ePrint Archive*, vol. 2007, p. 217, 2007.

[8] M. Green and G. Ateniese, "Identity-based proxy re-encryption," in *Applied Cryptography and Network Security*, 2007, pp. 288–306.

[9] C. Delerablée, "Identity-based broadcast encryption with constant size ciphertexts and private keys," in *ASIACRYPT'07*, 2007, pp. 200–215.

[10] B. Lynn *et al.* (2006). PBC library, [Online]. Available: https://crypto.stanford.edu/pbc/.

[11] T. Granlund *et al.*, *GMP, the GNU multiple precision arithmetic library*, 1991.

[12] R. S. Sandhu and P. Samarati, "Access control: Principle and practice," *IEEE communications magazine*, vol. 32, no. 9, pp. 40–48, 1994.

[13] T. Mayberry, E.-O. Blass, and A. H. Chan, "Efficient private file retrieval by combining ORAM and PIR," in *NDSS*, 2014.

[14] S. Devadas, M. van Dijk, C. W. Fletcher, L. Ren, E. Shi, and D. Wichs, "Onion ORAM: A constant bandwidth blowup oblivious RAM," in *Theory of Cryptography Conference*, Springer, 2016, pp. 145–174.

[15] D. Apon, J. Katz, E. Shi, and A. Thiruvengadam, "Verifiable oblivious storage," in *Public Key Cryptography*, 2014, pp. 131–148.

[16] T. Alves and D. Felton, "Trustzone: Integrated hardware and software security - enabling trusted computing in embedded systems.," in *Whitepaper*, 2004.

[17] V. Costan, I. A. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation.," in *USENIX Security Symposium*, 2016, pp. 857–874.

[18] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptology ePrint Archive*, vol. 2016, p. 86, 2016.

[19] Intel, *Intel software guard extensions SDK developer reference for Linux OS*, version 1.9, 2017. [Online]. Available: https://download.01.org/intel-sgx/linux-1.9/docs/Intel_SGX_SDK_Developer_Reference_Linux_1.9_Open_Source.pdf.

[20] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC: USENIX Association, 2017.

[21] N. Ferguson and B. Schneier, *Practical cryptography*. Wiley New York, 2003, vol. 23.

[22] D. R. Stinson, *Cryptography: theory and practice*. CRC press, 2005.

[23] C. Ellison and B. Schneier, "Ten risks of PKI: What you're not being told about public key infrastructure," *Computer*, vol. 16, no. 1, pp. 1–7, 2000.

[24] B. Waters, "Efficient identity-based encryption without random oracles," in *Eurocrypt*, Springer, vol. 3494, 2005, pp. 114–127.

[25] O. Weisse, V. Bertacco, and T. Austin, "Regaining lost cycles with HotCalls: A fast interface for SGX secure enclaves," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ACM, 2017, pp. 81–93.

[26] A. Fiat and M. Naor, "Broadcast encryption," in *Annual International Cryptology Conference*, Springer, 1993, pp. 480–491.

[27] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Crypto*, Springer, vol. 3621, 2005, pp. 258–275.

[28] C. Delerablée, P. Paillier, and D. Pointcheval, "Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys," *Pairing-Based Cryptography–Pairing 2007*, pp. 39–59, 2007.

[29] C. Delerablée, "Identity-based broadcast encryption with constant size ciphertexts and private keys," in *Advances in Cryptology – ASIACRYPT 2007*, K. Kurosawa, Ed. Springer, 2007, pp. 200–215. DOI: 10.1007/978-3-540-76900-2_12.

[30] Intel. (2017). Intel software guard extensions SSL, [Online]. Available: https://github.com/intel/intel-sgx-ssl.

[31] P. Schmidt *et al.* (2017). Linux kernel git revision history, [Online]. Available: https://www.kaggle.com/philschmidt/linux-kernel-git-revision-history.

[32] A. Sahai, B. Waters, *et al.*, "Fuzzy identity-based encryption," in *Eurocrypt*, Springer, vol. 3494, 2005, pp. 457–473.

[33] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Security and Privacy, 2007. SP'07. IEEE Symposium on*, IEEE, 2007, pp. 321–334.

[34] D. Boneh, X. Boyen, and E.-J. Goh, "Hierarchical identity based encryption with constant size ciphertext," in *EUROCRYPT'15*, 2005, pp. 440–456.

[35] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," *Theory of Cryptography*, pp. 253–273, 2011.

[36] B. A. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov, "Iron: Functional encryption using Intel SGX,"

[37] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 1, pp. 1–30, 2006.

[38] M. Green, S. Hohenberger, B. Waters, *et al.*, "Outsourcing the decryption of abe ciphertexts," in *USENIX Security Symposium*, vol. 2011, 2011.

[39] A. Sahai, H. Seyalioglu, and B. Waters, "Dynamic credentials and ciphertext delegation for attribute-based encryption," in *Advances in Cryptology–CRYPTO 2012*, Springer, 2012, pp. 199–217.

[40] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, "Multicast security: A taxonomy and some efficient constructions," in *INFOCOM'99*, IEEE, vol. 2, 1999, pp. 708–716.

[41] D. Wallner, E. Harder, and R. Agee, "Key management for multicast: Issues and architectures," Tech. Rep., 1999.

[42] M. Naor and B. Pinkas, "Efficient trace and revoke schemes," in *International Conference on Financial Cryptography*, 2000, pp. 1–20.

[43] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang, "Enabling security in cloud storage SLAs with CloudProof," in *USENIX Annual Technical Conference*, vol. 242, 2011, pp. 355–368.

[44] J. Li, C. Qin, P. P. Lee, and J. Li, "Rekeying for encrypted deduplication storage," in *DSN'16*, IEEE, 2016, pp. 618–629.

[45] F. Wang, J. Mickens, N. Zeldovich, and V. Vaikuntanathan, "Sieve: Cryptographically enforced access control for user data in untrusted clouds," in *NSDI*, 2016, pp. 611–626.

[46] J. Behl, T. Distler, and R. Kapitza, "Hybrids on steroids: SGX-based high performance BFT," in *Proceedings of the Twelfth European Conference on Computer Systems*, ser. EuroSys '17, Belgrade, Serbia: ACM, 2017, pp. 222–237. DOI: 10.1145/3064176.3064213.

[47] R. Pires, M. Pasin, P. Felber, and C. Fetzer, "Secure content-based routing using intel software guard extensions," in *Middleware '16*, Trento, Italy: ACM, 2016, 10:1–10:10. DOI: 10.1145/2988336.2988346.

[48] N. El Mrabet and M. Joye, *Guide to Pairing-Based Cryptography*. CRC Press, 2017.

APPENDIX A

This appendix details the mathematical implications of adapting IBBE to IBBE-SGX. In typical IBBE schemes, Trusted Authorities (TAs) only execute the operations of system setup and extracting user secret keys. In IBBE-SGX, however, the administrator agent executes all membership operations by executing them inside an SGX enclave.

The IBBE scheme [29] conceptually relies on the idea of bilinear maps. Notated as: $e(\cdot,\cdot) : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, a bilinear map is defined by using three cyclic groups of prime order p, imposing bilinearity and non-degeneracy. *El Mrabet et al.* [48] provide a thorough overview of bilinear maps usage within the cryptographic setting. Moreover, the IBBE scheme implies the public knowledge of a cryptographic hash function $\mathcal{H}$, that maps user identity strings to values in $\mathbb{Z}_p^*$.

A. System Setup

The initial operation is identical for IBBE and IBBE-SGX. The algorithm receives (λ, m) as input, where λ represents the security strength level of the cryptosystem, and m encapsulates the largest envisioned group size. The output consists of the Master Secret Key M_{SK} and the system Public Key PK. To build M_{SK}, the algorithm randomly picks $g \in \mathbb{G}_1$and$\gamma \in \mathbb{Z}_p^*$: $M_{SK} = (g, \gamma)$. To construct the PK, the algorithm computes $w = g^{\gamma}$ and $v = e(g, h)$, where $h \in \mathbb{G}_2$ was randomly picked: $PK = \left(w, v, h, h^{\gamma}, h^{\gamma^2}, ..., h^{\gamma^m}\right)$. The computational complexity of the system setup algorithm is linear to m.

B. User Key Extraction

The key extraction operation is identical for IBBE and IBBE-SGX. For a given user identity u, the operation makes use of M_{SK} and computes : $U_{SK} = g^{(\gamma+\mathcal{H}(u))^{-1}}$.

C. Encrypt Broadcast Key

The algorithm for constructing a broadcast key differs by considering the specific usage assumption. If for IBBE the algorithm has to rely on PK, for IBBE-SGX one can make use of M_{SK}. In both cases, the group broadcast key is randomly generated by choosing a random value $k \in \mathbb{Z}_p^*$ and computing:

$$b_k = v^k \tag{1}$$

A group broadcast ciphertext (C_1, C_2) is then constructed by:

$$C_1 = w^{-k} \tag{2}$$

$$C_2 = h^{k \cdot \prod\limits_{u \in S} (\gamma+\mathcal{H}(u))} \tag{3}$$

For IBBE, γ cannot be used directly for computing C_2. Instead, the computation is carried out with a polynomial expansion of the exponent that uses the public key elements:

$$C_2 = \left(\left(h^{\gamma^n}\right) \cdot \left(h^{\gamma^{n-1}}\right)^{\mathcal{E}_1} \cdot \left(h^{\gamma^{n-2}}\right)^{\mathcal{E}_2} \cdot ... \cdot \left(h^{\gamma}\right)^{\mathcal{E}_{n-1}}\right)^k \tag{4}$$

$$\begin{aligned} \mathcal{E}_1 &= \textstyle\sum_{u \in S} \mathcal{H}(u) \\ \mathcal{E}_2 &= \textstyle\sum_{u_1, u_2 \in S, u1 \neq u2} \mathcal{H}(u_1) \cdot \mathcal{H}(u_2) \\ &... \\ \mathcal{E}_{n-1} &= \textstyle\prod_{u \in S} \mathcal{H}(u) \end{aligned}$$

For IBBE, computing C_2 is bound by the computations of all $\mathcal{E}$, thus requires a quadratic number of operations $O\left(|S|^2\right)$. In the case of IBBE-SGX, having access to M_{SK} allows computing C_2 directly using Formula 3. It thus requires a linear number of operations.

Moreover, we augment the ciphertext values with C_3, which will prove useful for the subsequent operations:

$$C_3 = h^{\prod\limits_{u \in S} (\gamma+\mathcal{H}(u))} \tag{5}$$

Note that C_3 can be stored publicly as it can be computed entirely from PK.

D. Decrypt Broadcast Key

The decrypt operation is executed identically for IBBE and IBBE-SGX, and relies on PK. A user can make use of her secret key u_{SK} to compute b_k, given $(\mathcal{S}, \mathcal{C})$ We chose to omit presenting the intricate formula as we maintain it in the original form, as shown in [29].

E. Add User to Broadcast Key

As the joining user u_{add} is allowed to decrypt group secrets prior to joining, there is no need of a re-key operation by changing the value of b_k. The only required change is therefore to incorporate u_{add} into $\mathcal{S}$, and $\mathcal{H}(u_{add})$ into C_2.

For IBBE, including $\mathcal{H}(u_{add})$ into all $\mathcal{E}$ values requires a quadratic number of operations. For IBBE-SGX, by making use of M_{SK}, one has access to γ, thus the new user is included in constant time: $C_2 \leftarrow (C_2)^{\gamma+\mathcal{H}(u_{add})}$.

F. Remove User form Broadcast Key

Whenever removing a user u_{rem}, all group elements $b_k, \mathcal{S}$ and $\mathcal{C}$ need to change. b_k and C_1 can be computed by Formulas 1 and 2, once a new random value for $k \in \mathbb{Z}_p^*$ is picked.

Within the traditional assumption, C_2 is computed similarly to encrypting group key operation, consuming a quadratic number of operations. Within IBBE-SGX, having access to γ through the M_{SK} allows first changing C_3 and then C_2 in constant time:

$$C_3 \leftarrow (C_3)^{(\gamma+\mathcal{H}(u_{rem}))^{-1}} \tag{6}$$

$$C_2 \leftarrow (C_3)^k \tag{7}$$

G. Re-key Broadcast Key

Sometimes, it is necessary to change the value of b_k without performing any group membership changes. This re-keying operation can be performed optimally in constant time under both usage model assumptions, by making use of C_3.

First, a new random $k \in \mathbb{Z}_p^*$ is generated and the new key computed by Formula 1. C_1 can be computed by Formula 2, while C_2 is computed from C_3 by Formula 7.