Anonymous and Confidential File Sharing over Untrusted Clouds

Stefan Contiu*‡, Sébastien Vaucher†, Rafael Pires†, Marcelo Pasin†, Pascal Felber† and Laurent Réveillère*
*University of Bordeaux, France, firstname.lastname@u-bordeaux.fr (‡Scille SAS, France)
†University of Neuchâtel, Switzerland, firstname.lastname@unine.ch

***Abstract*—Using public cloud services for storing and sharing confidential data requires end users to cryptographically protect both the data and the access to the data. In some cases, the identity of end users needs to remain confidential against the cloud provider and fellow users accessing the data. As such, the underlying cryptographic access control mechanism needs to ensure the anonymity of both data producers and consumers.**

We introduce A-SKY, a cryptographic access control extension capable of providing confidentiality and anonymity guarantees, all while efficiently scaling to large organizations. A-SKY leverages *trusted execution environments* (TEEs) to address the impracticality of *anonymous broadcast encryption* (ANOBE) schemes, achieving faster execution times and shorter ciphertexts. The innovative design of A-SKY limits the usage of the TEE to the narrow set of data producing operations, and thus optimizes the dominant data consumption actions by not requiring a TEE. Furthermore, we propose a scalable implementation for A-SKY leveraging micro-services that preserves strong security guarantees while being able to efficiently manage realistic large user bases. Results highlight that the A-SKY cryptographic scheme is 3 orders of magnitude better than state of the art ANOBE, and an end-to-end system encapsulating A-SKY can elastically scale to support groups of 10 000 users while maintaining processing costs below 1 second.

I. INTRODUCTION

Relying on cloud services for storing content emerges as an efficient method for organizations to cut and adapt functional costs. As cloud service providers cannot be fully trusted [1], data owners shall cryptographically protect their data before sending it to storage providers, by encrypting it with secret keys. Furthermore, data owners grant access to well-defined groups of users to create and consume that data. Due to lack of trust in cloud providers, cryptographic access control mechanisms are used instead to store and enforce that only valid users can access the keys and, consequently, the data.

Sometimes, not only data but also the identity of users is sensitive and has to be protected. Consider for example military organizations that define access groups based on security clearances. Besides protecting the shared information that is specific to a clearance level (*e.g.*, confidential, secret and top secret), users sharing the same clearance level do not know each other. Likewise, dispatching confidential medical programs (*e.g.*, for HIV patients) needs to ensure that patients' privacy is guaranteed [2] and therefore fellow patients cannot infer their identity. Moreover, Virtual data rooms (VDRs) [3] used for exchanging confidential documents during business acquisitions not only need to enforce a high-level of access control, but also to protect stakeholders' identities.

Existing research in the area of security of cloud-backed storage systems covers cryptographic access control for data confidentiality and authenticity [4]–[6], but not anonymity. These systems rely on public key cryptography mapping user identities and enveloping symmetric keys that protect the actual shared content. Differently, confidential systems focusing on group communication offer anonymity guarantees by group key exchange methods [7], requiring all active group members to be present and participate in a multi-phase protocol (*e.g.*, Diffie-Hellman) each time a key is derived. Such an approach is indeed suitable for instant group communication, but impractical for file sharing that generally does not require the online presence of users. Moreover, theoretical anonymous file sharing extensions have been hypothesized [8], [9] without ever turning into functional systems. The need for anonymous sharing of confidential content was practically addressed in an unsophisticated manner by GNU privacy guard (GPG). The approach implemented by GPG is to drop any public key mapping from the resulting ciphertext, and therefore keep no reference to the identity of the actual content's recipients. The main drawback of this solution happens at decryption time, when the recipient needs to perform many asymmetric decryption trials until the portion of the ciphertext matching his private key is found (if any). As pointed by our preliminary benchmark (§III, Table I), GPG works well for groups of few users but quickly becomes impractical for larger ones.

As an alternative approach, trusted execution environments (TEEs) such as Intel Software guard extensions (SGX) [10] or ARM TrustZone [11] have seen rapid adoption during the last few years. Data and computations happening within such trusted environments cannot be seen from outside. A number of security systems profited from TEE integration in order to achieve practical performance while targeting strict threat models [12], [13]. Envisioning TEE usage as a building block for anonymous sharing systems is therefore natural. However, TEEs and more specifically Intel SGX come with side costs, notably due to transitioning latency between trusted and untrusted zones, as well as page swapping when exceeding the limited memory size of the enclave page cache (EPC). In addition, one cannot rely on widespread adoption of such enabling technology. Instead, one needs to consider the participants' heterogeneity in an anonymous sharing scheme, including various microprocessor architectures, mobile users or even Internet of things (IoT) devices.

In this paper, we propose an anonymous access control scheme that leverages SGX as TEE only for a narrow scope and deployment: enforcing anonymity during the publishing operation (*i.e.*, upon *writing*). Our scheme does not require a TEE on the user side for performing the *read* operation, nor does it require that users connect to a designated TEE proxy. Moreover, by leveraging TEEs, we can circumvent assumptions of state-of-the-art theoretical anonymous sharing schemes [8] and considerably improve the performance of

2575-8462/19/$31.00 ©2019 IEEE
DOI 10.1109/SRDS47363.2019.00013

cryptographic operations. To demonstrate the feasibility of our solution, we propose a scalable system design leveraging micro-services that can elastically scale depending on the access control and data content workloads.

Even though our work targets file sharing over untrusted cloud storages, the proposed solution can be adapted to a wider spectrum of anonymous broadcast contexts such as media streaming or peer-to-peer networks.

Our evaluation highlights that our construct is faster by 3 orders of magnitude compared to state-of-the-art anonymous broadcast encryption (ANOBE) [8]. Furthermore, our end-to-end system implementation, A-SKY, can adequately scale to cope with a similar number of administrative and user operations that a realistically-sized organization would experience (see §VI).

In short, we propose the following original contributions: *(i)* We define a theoretical anonymous cryptographic access control extension that relies on TEEs for a minimal subset of operations (*i.e.*, *writes* but not *reads*). To the best of our knowledge, our approach is the first to leverage TEEs for the construction of ANOBE primitives. *(ii)* We propose an end-to-end system design, incorporating our theoretical construct and leveraging micro-services that can scale according to the undergoing workloads. *(iii)* We implement and evaluate the system, first in isolation showing its benefits against state-of-the-art cryptographic schemes, and secondly by benchmarking its scaling capabilities and practical feasibility.

The paper continues by introducing the actors and adversarial threat model (§II). We then discuss the state of the art and open challenges (§III), present the design of our solution (§IV) and its implementation (§V), evaluate our prototype within isolated micro-benchmarks and large-scale macro-benchmarks (§VI), and finally conclude (§VII).

II. MOTIVATION

We provide an overview of the assumptions and security objectives of file sharing systems that guarantee data confidentiality and user anonymity.

1) Model and Use Case: We target file sharing between *users* represented by humans or software agents. We consider that users are uniquely identified within the premises of an organization. Users are organized into uniquely identifiable *groups* by organization-specific considerations and policies. We consider a separation between the group access control and group content management by both functional and threat factors. Group access control represents group memberships operations and is performed by *administrators*. Administrative operations consist in adding and removing users from *groups*. Group content management represents creating and consuming files by group members. A user can hold one or both roles of *writer* and *reader* within one or multiple groups. The remote storage is a typical cloud object storage that can store uniquely-identified large binary objects (*e.g.*, Amazon S3).

Exemplifying Use Case. Virtual data rooms (VDRs) [3] enable a tightly controlled exchange repository of electronic documents for company mergers and acquisitions (M&A). Thanks to VDRs, the seller, supporting parties assisting the seller, and acquisition bidders can confidentially exchange documents (*e.g.*, terms, valuation) through an untrusted remote storage medium. The seller acts as *administrator* and enforces access control. *Active* user roles are constituted by *writers* (the seller and supporting parties) and *readers* (the bidders). As enforced by confidentiality agreements, supporting parties operate under the umbrella of the seller, and remain unidentifiable from each other. Similarly, the seller can conceal the identity of bidders among themselves. As such, *inner* anonymity guarantees need to be enforced within the *writers* (supporting parties) and *readers* (bidders) groups, while *outer* anonymity needs to withstand against any actor who is not involved in the M&A process. Additionally, any withdrawing bidder or misbehaving supporting party can be *revoked* by the seller, and therefore unable to access the document corpus. The operation load of such scenario follows typical workloads of cloud sharing services (*e.g.*, Dropbox, Google Drive), modeled by YCSB [14] in our macro-benchmark evaluation (§VI-2).

2) Security Objectives: We specify four high-level security properties for confidential and anonymous file sharing systems.

(i) **Confidentiality and Authenticity:** The secrecy of the content of shared files is exclusive for the group members. Recipients should be able to check the integrity and provenance of shared content.

(ii) **Forward Secrecy:** The compromise of a group secret should not compromise past sharing sessions within the same group.

(iii) **Recipients Privacy:** No recipient except the group administrator should be able to infer the identities of other recipients (*i.e.*, *readers*).

(iv) **Sender Privacy:** No recipient except the group administrator should be able to infer the sender's (*i.e.*, *writer*) identity.

3) Threats: Revoked users and users external to the system behave arbitrarily. They try to discover shared content and group members identities. To do so, they can intercept, decipher and alter exchanged messages (*i.e.*, Dolev-Yao [15] adversarial model).

User anonymity is not only endangered by external adversaries, but also internally by considering peer group members. As such, we consider that active users that can rightfully decrypt group content are able to launch attacks with the goal of inferring peer members identities. To do so, they can make use of unlimited attack trials *adapted* to their adversarial strategy. We therefore consider that the proposed solution should satisfy the strong security notion of *adaptive chosen ciphertext attack* (IND-CCA2). A solution that fulfills such guarantees also satisfies weaker security notions of non-adaptive chosen ciphertext or plaintext attacks.

The storage provider behaves in an *honest-but-curious* manner. As such, it can try to observe the incoming and outgoing data flows with the goal of discovering the actual content and the identity of the users accessing the data, all while providing service. In order to break the confidentiality guarantee, revoked users can collude with the cloud storage to discover content created after their revocation.

Finally, our privacy model enforces the anonymity guarantee only with respect to user identities. We consider hiding the size of groups, how often members communicate and the size of the content that they exchange as out-of-scope.

III. Related Work

This section discusses related work and open challenges in the domain of cryptographic cloud storage and access control, as well as trusted execution environments.

Cryptographic Cloud Storages and Access Control. In recent years, a number of storage and sharing system designs have been proposed for mitigating the lack of trust in cloud providers. DepSKY [16] proposes an object store interface that can be used on the client side to encrypt and redundantly store ciphertext on multiple untrusted storages. The encryption keys are split by using a secret sharing scheme [17] and dispersed over multiple storage systems that do not collude with each other. SCFS [1] extends the client-side encryption and cloud redundancy of DepSKY by using a trusted metadata coordination service that also encapsulates access control.

Some systems follow a different avenue by *cryptographically* enforcing access control using key enveloping. Also referred to as *hybrid encryption* [4], the technique consists in encrypting data with a symmetric key that is then itself encrypted with public key encryption. For example, CloudProof [5] proposes a client-side encrypted cloud storage that solves access control by using *broadcast encryption* [18] to envelope two keys: the first is used for decrypting (*i.e.*, *reads*) and the second one for signing (*i.e.*, *writes*). Differently, REED [19] uses *attribute-based encryption* [20] to envelope the symmetric keys that are protecting de-duplicated content. However, the key-enveloping technique was argued by Garrison *et al.* [4] as impractical when target usage conditions are highly dynamic. IBBE-SGX [6] demonstrates that the approach can be implemented within dynamic conditions when leveraging trusted execution environments (TEEs). Yet, none of the above constructions considers an enriched threat model for preserving both confidentiality and anonymity.

Confidential Messaging Systems. Encrypted messaging systems share a common initial phase with our cloud file sharing model, by requiring the construction of a group key that protects group communication. Popular messaging systems (*e.g.*, WhatsApp, Threema, Signal) use a Diffie-Hellman (DH) group key agreement and derivation [21]. Such protocols require all active participants to contribute to the creation of the group key, albeit without providing anonymity guarantees. Pung [7] uses private information retrieval (PIR) in conjunction to a group DH key derivation, thus achieving anonymity. Such a mechanism is different from our target model, in which active users do not need to participate in the creation of the group key, no matter the number of groups they belong to.

Pretty Good Privacy. In practical systems, the popular Pretty good privacy (PGP) [22] program, used for cryptographic protection of file or emails, addressed the anonymity criteria with a simple solution. In anonymous mode (or *hidden recipient* as called by PGP), after performing the symmetric encryption of the content and public key encryptions of the symmetric key, all the public key mappings are dropped from the resulting ciphertext. As such, an outside adversary cannot infer the public keys of the recipients. At decryption time, as the recipients have no pointer to their key-envelope ciphertext fragment, they need to perform several private key decryption trials until they succeed ($\frac{n}{2}$ trials on average, where n is the group size). Table I presents results of a simple benchmark of GPG (v. 1.4.2) in *hidden recipient* mode. One can observe that encryption and—even more notably—decryption have an impractical cost of 12 and 60 s respectively for groups of 1000 members. Moreover, the inner implementation of PGP's *hidden recipient* mode is reputed as insecure against chosen ciphertext attacks [8], our targeted threat model.

TABLE I: GPG operations latency in *hidden recipient* mode.

Group size	Encrypt [s]	Avg. decrypt [s]	Size [kB]
10	0.13	0.6	5.3
10^2	0.7	5.8	16.5
10^3	12	60	129

Anonymous Broadcast Encryption. The theoretical problem of devising a cryptographic scheme that can guarantee both confidentiality and anonymity is referred to as *anonymous* (or *private*) *broadcast encryption* (ANOBE). Theoretical research literature proposes a number of such schemes, however without assessing their practicality within real systems.

The *private broadcast encryption* proposed by Barth *et al.* [8] (denoted hereafter *BBW*, per the authors' initials) achieves inner and outer anonymity, in addition to providing IND-CCA guarantees. Their construction extends the public key enveloping model of PGP, by incorporating strongly unforgeable signatures [23] such that an active attacker who is member of the group cannot reuse the envelope to broadcast arbitrary messages to the group. Moreover, to decrease the number of decryption trials, they propose the construction of publicly-known labels, unique for each member of every single encryption operation, by relying on the security assumption of DH. The ciphertext fragments created by the key enveloping process are therefore ordered by their label. During decryption, after reconstructing the label, the user can seek the corresponding ciphertext fragment in logarithmic time before performing a single asymmetric decryption. The scheme was further extended by Libert *et al.* [9] by suggesting the use of *tag-based encryption* [24] to hint users to their ciphertext fragment. To the best of our knowledge, no practical system has integrated *tag-based encryption* in practice.

As pointed out by our comparison benchmark (§VI-1), *BBW* [8] can handle a key enveloping throughput of only few hundreds of users per second. Such a limitation requires the exploration of alternative constructions that can scale to realistic access control workloads.

Trusted Execution Environments. Recently, TEEs gathered considerable interest as an approach for solving the otherwise difficult problem of securely hosting services in the cloud, while making sure that the infrastructure provider has no knowledge of the data it handles. Examples include performing map reduce computations [12], machine learning algorithms [25] or analytics [26], while offering confidentiality guarantees to end users.

A popular choice of TEE technology is Intel Software guard extensions (SGX). It defines the concept of *enclave* as an isolated unit of data and code execution that cannot be accessed even by privileged code (*e.g.*, the operating system) [10]. Enclaves can be *attested*, that is, proving that the code that runs is the one intended, and that it is running on a genuine Intel SGX platform. It seems therefore natural to rely on SGX as building block for an anonymous sharing system. A *naïve*

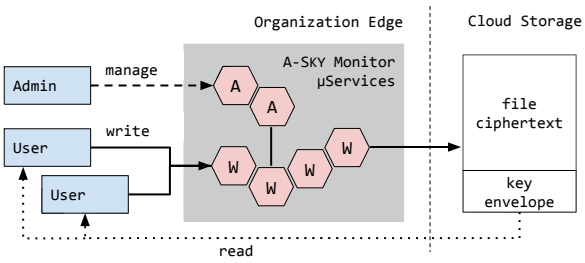

Fig. 1: A-SKY solution overview. A-SKY monitor services are ACCESSCONTROL (**A**) and WRITERSHIELD (**W**).

approach would be to require end users to use this enabling technology and perform any access control related operations in full isolation. We argue that this approach is impractical due to the heterogeneity of end users computing platforms, which might not be necessarily equipped with SGX capabilities.

A different approach that makes use of TEEs is to proxy all the access control, and therefore the *read* and *write* operations, through a broker service that runs within enclaves. However, we claim that given the memory and computational limitations of SGX enclaves (*e.g.*, trusted computing base (TCB) size, trusted/untrusted transition latency), it is far from trivial to develop such a proxy service able to scale and sustain a high data throughput, considering dynamic access control operations [4]. While achieving scale-out by micro-services that run on top of Intel SGX is possible in a containerized environment [27], our challenge is to define an optimal architecture for an anonymous sharing system that incorporates SGX as TEE with minimal performance overhead.

IV. A-SKY

Our solution conceptually relies on two paradigms: a cryptographic key management solution and a data delivery protocol, both designed to target an increased system performance, covering data confidentiality and user anonymity guarantees. We describe our solution by first having an overall look into the proposed architecture. We continue by detailing the design of each system operation. Finally, we briefly discuss the security guarantees of our scheme.

1) Architectural Overview: A-SKY leverages Intel SGX as a TEE. In order to avoid passing all the system operations through a TEE-enabled *monitor*, we propose a design in which only data owners (*i.e.*, *writers*) are constrained to pass through such a proxy. *Readers* anonymously consume confidential content without needing to pass through the TEE-enabled monitor, therefore not incurring in service time penalties. The benefits of using a monitor exclusively for write operations are manifold. First, the monitor acts as an outbound trusted authority (TA) authenticating all the content passing through. Second, it can mask the identities of data writers. Third, as the monitor executes in a TEE, traditional anonymous key management schemes [8], [9] can be modified to accommodate a new entity of trust for the key enveloping operation, therefore allowing more efficient operations.

Fig. 1 displays the overview of A-SKY solution. The A-SKY monitor sits in between end-users and the cloud storage and is logically split in two roles. First, it provides a cryptographic mechanism for storing and enforcing access control to the data, by offering a cryptographic key management solution (the ACCESSCONTROL service). Second, upon successful access verification, it acts as an outgoing proxy for write operations (the WRITERSHIELD service). As system scalability is of paramount importance, the two logical entities (ACCESSCONTROL and WRITERSHIELD services) can independently adapt to undergoing load.

Key Management. The first building block of our design is a cryptographic key management solution. Data owners have to write content through the TEE-enabled monitor such that only authorized readers (who are not passing through the TEE monitor) can decrypt the data, all while having anonymity guarantees. In traditional anonymous key sharing solutions [8], a TA performs two operations: setting up the key management system and extracting user private keys. The operations of key enveloping together with content encryption and decryption are performed by end users. As such, public key cryptographic primitives are employed so that end users can cryptographically protect content for other users, whose identities are represented by public keys. Differently, our model that leverages a TEE as an outgoing monitor requires that the TA does not only set up the system and extracts user keys, but also executes the key enveloping operation, which in the traditional assumptions was executed by end users. This change of assumption therefore allows us to use a much simpler cryptographic construct to achieve the same result as traditional schemes. Concretely, the TA can directly make use of users' secret keys during the key-enveloping operation. As such, this shared secret between the TA and end users opens the way to the use of *symmetric* rather then *public key* cryptography, and therefore benefit from the performance advantages of the former, *e.g.*, hardware acceleration and smaller ciphertexts. Second, as the traditional scheme [8] requires the construction of a signature key-pair per each key enveloping, under the new assumptions we can leverage the signature of the TA. Moreover, the shared secret between users and TA allows to construct efficient key de-enveloping methods that increase the performance of the decryption operation performed by end users.

Data Delivery Protocol. A-SKY allows users to write the encrypted shared content through the WRITERSHIELD service, which acts as a proxy. The service will check with the ACCESSCONTROL whether a user is granted the permission to write in a given group. Being the case, it authenticates the outgoing content and does the writing itself. We can therefore securely store the cloud storage credentials in the WRITERSHIELD service.

2) TEE Trust Establishment: Before relying on any service of the A-SKY *monitor*, it is necessary to validate that the service is running on a trustworthy Intel SGX platform, and that the instances of the ACCESSCONTROL and WRITERSHIELD services are genuine. This validation phase is performed by *administrators*, who are considered fully trusted (see §II).

As such, the SGX enclaves are required to construct a proof that incorporates the digest of the code and data inside the enclave, signed by the fused CPU private key (also known as *quote* [10]) and whose corresponding public key is retained by Intel. The attestation packages are retrieved by the *administrators*, who in turn check that the received digests are identical to known ACCESSCONTROL or WRITERSHIELD digests. They

Algorithm 1 Key enveloping (ACCESSCONTROL)

Input: user identity u_{id}, group identifier g_{id}, symmetric key k.
Output: an $envelope$ ciphertext of the access control key.
1: $envelope \leftarrow \emptyset$
2: **if** $u_{id} \in group^w[g_{id}]$ **then**
3: **for all** users $u \in group^r[g_{id}]$ **do**
4: $u_{sk} \leftarrow keys[u]$
5: $(c_k,t) \leftarrow AE_{u_{sk}}(k)$
6: $envelope \leftarrow envelope \cup \{(c_k,t)\}$
7: **end for**
8: **end if**
9: **return** $envelope$

then contact Intel's *remote attestation service* to validate that the quote signature is indeed genuine. Upon a successful verification, *administrators* rely on the remote attestation functionality to establish a secure channel using a DH key exchange with both the ACCESSCONTROL and WRITERSHIELD services [10]. This secure channel is used for subsequent access control operations, such as user creation, addition or removal of members from groups. Besides, administrators are able to securely provide the cloud storage credentials to the WRITERSHIELD service along with a long term signing key $sign_{TA}$ that is employed on all upcoming transiting content.

3) Operations Design: This subsection formally defines the operations of the ACCESSCONTROL and WRITERSHIELD services.

Let $E_k(p) \rightarrow c$ and $D_k(c) \rightarrow p$ define symmetric encryption and decryption algorithms using the key k, where p is plaintext and c is ciphertext. We denote by $AE_k(p) \rightarrow (c,t)$ and $AD_k(c,t) \rightarrow \{p,\perp\}$ authenticated encryption and decryption algorithms that, besides the above symmetric primitives, can produce an authentication tag t proving the integrity of the ciphertext c under the key k. We denote by $S_{pri}(p) \rightarrow \sigma$ and $V_{pub}(p,\sigma) \rightarrow \{true,\perp\}$ digital signature and verification schemes employing an asymmetric key-pair (pri and pub). Finally, $\mathcal{H}$ denotes a one way cryptographic function and $||$ denotes the literal concatenation operation.

ACCESSCONTROL. The ACCESSCONTROL service is responsible for storing credentials, membership information and to enforce them. Its methods are invoked by administrators through the secure channel established upon successfully performing the trust attestation process (§IV-2).

The ACCESSCONTROL service generates user secret keys. Given a unique user identifier u, the service constructs a random secret key for the user, to whom it is sent through a transport layer security (TLS) channel.

The ACCESSCONTROL service further exposes methods for group management. Specifically, administrators can create groups, as well as add or remove users from groups. Depending on the granted access capabilities, users can hold the roles of content *reader*, *writer*, or both. The ACCESSCONTROL service captures such capabilities within persistent dictionaries, $group^r$ and $group^w$, which store lists of users belonging to each group identifier (*e.g.*, $group^w[g_{id}] = \{u_a, ..., u_z\}$). Administrators are the only entities that can modify the keys and values of those two dictionaries.

The operation of enveloping an access key for a group of anonymous members is denoted by *KeyEnveloping* and is depicted in Alg. 1. Given the identity of the writing user, the group unique identifier, and a symmetric key k, the algorithm produces a ciphertext *envelope* that can be anonymously de-enveloped. The operation proceeds by first checking that the user has writing capabilities for the group (line 2). If true, the *envelope* is constructed by including the ciphertext and the authentication tag resulted from encrypting the symmetric key using the secret key of each group member (lines 3-7).

Algorithm 2 Proxy file (WRITERSHIELD)

Input: user identity u_{id}, group identifier g_{id}, file ciphertext $\mathcal{C}$, ACCESSCONTROL instance $\mathcal{A}$.
1: **if** $u_{id} \in \mathcal{A}.group^w[g_{id}]$ **then**
2: $\sigma \leftarrow S_{sign_{TA}}(\mathcal{C})$
3: $Upload$ to cloud : $(\mathcal{C},\sigma)$
4: **end if**

Algorithm 3 User write file to group

Input: user identity u_{id}, group identifier g_{id}, file plaintext P, ACCESSCONTROL and WRITERSHIELD instances $\mathcal{A}$ and $\mathcal{W}$.
1: $fk \leftarrow Random$ symmetric key
2: $envelope \leftarrow \mathcal{A}.KeyEnveloping(u_{id},g_{id},fk)$ *i.e.*, Alg. 1
3: $cipher \leftarrow E_{fk}(P)$
4: $\mathcal{C} \leftarrow envelope \;||\; cipher$
5: $\mathcal{W}.ProxyFile(u_{id},g_{id},\mathcal{C},\mathcal{A})$ *i.e.*, Alg. 2

Algorithm 4 User read file

Input: user secret key u_{sk}.
1: $Download$ from cloud: $(\mathcal{C},\sigma)$
2: **if** $V_{pub-sign_{TA}}(\mathcal{C},\sigma) \neq \perp$ **then**
3: $envelope, cipher \leftarrow split(\mathcal{C})$
4: **for all** pairs (k_c,t) in $envelope$ **do**
5: $fk \leftarrow AD_{u_{sk}}(k_c,t)$
6: **if** $fk \neq \perp$ **then**
7: $P \leftarrow D_{fk}(cipher)$
8: **return** P
9: **end if**
10: **end for**
11: **end if**
12: **return** $\perp$

WRITERSHIELD. As the WRITERSHIELD is the sole service possessing the write credentials for the cloud provider, it constitutes a necessary hop for uploading the file. Its main operation is $ProxyFile$ (Alg. 2). The method verifies that the invoking user has write capabilities for the desired group (line 1). If positive, the content is authenticated by using the long term TA signature (line 2). Both parts, file ciphertext and the corresponding signature, are finally uploaded to the cloud (line 3).

USER. The two operations performed by users are sharing a file with a group (*i.e.*, writing) and reading a shared file. The user write operation leverages the TEE-enabled monitor. As shown in Alg. 3, the user first randomly creates a symmetric key (line 1) and asks the ACCESSCONTROL service to perform an enveloping for this key (line 2), so that it can be anonymously de-enveloped by any group member. He then encrypts the file by using the prior generated symmetric key (line 3). Finally, the two obtained ciphertexts—the key envelope and the file ciphertext—are concatenated (line 4) and transmitted to the WRITERSHIELD to be uploaded to the cloud storage (line 5).

Users can read files by following the procedure of Alg. 4. As previously stated, reading operations do not involve services running in a TEE. The first step is to download the

Algorithm 5 Key enveloping with *efficient decryption*

Input: user identity u_{id}, group identifier g_{id}, symmetric key k.
Output: an $envelope$ ciphertext of the access control key.
1: $envelope \leftarrow \emptyset$
2: **if** $u_{id} \in group^w[g_{id}]$ **then**
3: $nonce \leftarrow Random$
4: **for all** users $u \in group^r[g_{id}]$ **do**
5: $u_{sk} \leftarrow keys[u]$
6: $l_u \leftarrow \mathcal{H}(u_{sk} \mid\mid nonce)$
7: $(c_k,t) \leftarrow AE_{u_{sk}}(k)$
8: $envelope \leftarrow envelope \cup \{(l_u,c_k,t)\}$
9: **end for**
10: $Sort$ $envelope$ by l (*i.e.*, $label$)
11: **end if**
12: **return** $nonce \mid\mid envelope$

ciphertext package from the cloud storage (line 1), that can then be validated by checking the signature (line 2) that has been appended by the WRITERSHIELD. Should the signature be valid, the user then splits the package between the key envelope and the file ciphertext (line 3). Next, the user iterates over all envelope fragments, trying to decrypt each of them by using the user secret key u_{sk} (lines 4-5). If successful, the obtained plaintext is the file encryption key, that the user can use to symmetrically decrypt the file ciphertext (lines 7-8).

4) Indexing for Efficient Decryption: Following the methodology of traditional ANOBE schemes [8], [9], we propose a method that can reduce the user decryption time by circumventing the need to perform several key decryption trials (line 4 of Alg. 4) by trading it off for a slight increase in key enveloping time and envelope size. To this end, publicly known labels are constructed for each user fragment in the envelope, such that the label can be recomputed by the target recipients. User keys are ordered by labels in the envelope, so that each key can be easily located within it and a single key decryption operation is performed. Traditionally, the cost of building such labels was associated to performing modular exponentiation [8] or by using the theoretical constructs of tag-based encryption [9]. Given the change of assumption brought by A-SKY compared to traditional ANOBE, we now have a TA running in a TEE performing the key enveloping. It results that the shared secret between users and the TA can also be used to construct efficient decryption labels. A-SKY can therefore propose a much simpler and efficient labeling mechanism by relying on the cryptographic hash of the shared secret (*i.e.*, the user secret key).

The efficient variant of key enveloping (Alg. 5) introduces the creation of labels (line 6) as the salted hash of the user secret key. A random *nonce* is generated for each key enveloping call to be used as a salt value, publicly included in the envelope. The envelope fragments can therefore be sorted using the label values (line 10).

A user read operation (Alg. 6) requires the label reconstruction (line 4) followed by a binary search of it among the envelope fragments (line 5). When the proper label is located, the file key can be retrieved (line 7), allowing at last the file decryption (lines 8–9).

The trade-off brought by this *efficient decryption* method is therefore an overhead of $O(n \cdot \log n)$, due to the sorting of the labels during the key enveloping operation. The gain is reflected during decryption time, replacing $O(n)$ trials of symmetric decryption with a $O(\log n)$ binary search and a single symmetric decryption.

Algorithm 6 User read file with *efficient decryption*

Input: user secret key u_{sk}.
1: $Download$ from cloud : $(\mathcal{C},\sigma)$
2: **if** $V_{pub-sign_{TA}}(\mathcal{C},\sigma) \neq \perp$ **then**
3: $nonce$, $envelope$, $cipher \leftarrow split(\mathcal{C})$
4: $l_u \leftarrow \mathcal{H}(u_{sk} \mid\mid nonce)$
5: $(k_c,t) \leftarrow$ *binary search* for key: l_u in $envelope$
6: **if** $(k_c,t) \neq \perp$ **then**
7: $fk \leftarrow AD_{u_{sk}}(k_c,t)$
8: $P \leftarrow D_{fk}(cipher)$
9: **return** P
10: **end if**
11: **end if**
12: **return** $\perp$

5) A Note on Revocation: We argue that A-SKY satisfies the *lazy* revocation model [28], where a revoked user can continue accessing data created prior to revocation but should be unable to access any data created beyond that. Additionally, past data becomes inaccessible upon the first succeeding write to the same resource.

The revocation is triggered by an administrator removing the user's id from the $group^r$ and $group^w$ access lists. Later, when new content is published in that group, a new random key is derived for encrypting the content (Alg. 3, line 1), and a new envelope is attached to it (Alg. 3, line 2). The revoked user's key will not be included in the envelope, and therefore the user will be unable to access the new group key along with the newly published content.

V. IMPLEMENTATION

1) ACCESSCONTROL*:* The ACCESSCONTROL service is the only stateful component of A-SKY. It is responsible for generating and storing user keys, and for maintaining group membership information. Since it deals with sensitive information, its core runs entirely within enclaves. All external exchanges are encrypted by using TLS connections that are terminated inside trusted environments.

We divide the ACCESSCONTROL service into two subcomponents. The first one constitutes the entry-point for service requests. It is developed in C++, for a total of 3000 lines of code (LoCs).

The other one holds users and groups metadata within a replicated database. For this purpose, we use a triple-replicated cluster of MongoDB [29] servers. MongoDB offers out-of-the-box scale-out support, and is well suited to store denormalized documents. In order to perform queries against it from the first sub-component, we ported the official MongoDB client library [30] to run inside an enclave. Each replica of the entry-point sub-component is provisioned with the master key M_k at *attestation* time; its purpose is to secure the data stored in the MongoDB backend.

As the storage backend runs outside of enclaves, we make sure that every piece of data that we store is either hashed using the HMAC-SHA256 construct or encrypted using advanced encryption standard (AES) Galois counter mode (GCM). We thus guarantee that the entity that provides the MongoDB instances cannot infer any information about users or groups (barring the size of each group, which is already leaked in the *envelopes*). Fig. 2 shows how we organize data

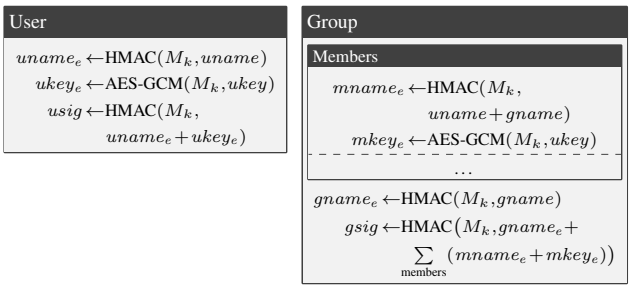

Fig. 2: Data model of user and group *documents* stored in MongoDB.

in MongoDB. We use 2 collections, one for users and one for groups. Each user is stored once in the users collection and once per group it is a member of. This denormalization prevents the service provider from inferring which groups a user belongs to as the attributes of a given user are hashed or encrypted differently for each representation (*i.e.*, we use the name of the group as salt when hashing, and different initialization vectors (IVs) when encrypting). Each document is wholly signed using HMAC signatures to ensure its integrity.

There are two kinds of users interested in communicating with the AccessControl service: regular users, who need to retrieve their randomly-generated $256\,\mathrm{bit}$ private key, and administrators, who perform group access control operations. All these interactions happen through a TLS-encrypted REST-like protocol. Exchanges are represented in JavaScript object notation (JSON), for which we slightly modified a C++ library [31]. In order to terminate TLS connections in the enclave, we use a port of OpenSSL for SGX [32].

Another duty of the AccessControl service is to generate key envelopes upon user requests. An envelope contains a file key encrypted several times, once per group member. The file key, as well as the user keys, are $32\,\mathrm{B}$ long. We use AES GCM, which generates a *tag* of $16\,\mathrm{B}$ for integrity. Considering the addition of $12\,\mathrm{B}$ for the IV, each group member adds $60\,\mathrm{B}$ to the envelope.

In order to avoid having to perform $O(n)$ decryption trials, we can index the keys within the envelope (§IV-4). First, we generate a *nonce* for each envelope, that we staple to it. Each user key is then hashed using the SHA224 algorithm, using the *nonce* as a salt. This adds $28\,\mathrm{B}$ to the envelope for each group member. The list of keys is sorted using the hashes as a sorting key. As a consequence, *readers* can look for their own key by doing a binary search, therefore decreasing the complexity to $O(\log n)$ comparisons followed by one single decryption. We evaluate the trade-offs as far as enveloping time and bandwidth usage are concerned in §VI.

2) WriterShield*:* The WriterShield serves two purposes: protecting cloud storage credentials, and hiding user identities by proxying their requests to the cloud storage. When forwarding user requests to write files, the WriterShield checks with the AccessControl that the query comes from a user who has the correct permissions to write files. User requests, including file contents, cross over the enclave boundary. This obviously slows down transmission rates because of content re-encryption and trusted/untrusted edge transitions. Therefore, we have also implemented a different variant where temporary access tokens are given to users, allowing them to upload their content without the aforementioned content needing to enter the TEE. Note that the ciphertext digest still needs to be authenticated by the signing key available in the TEE-enabled service (necessary for IND-CCA2). In such case, users are responsible for using appropriate proxies that can conceal the origin of the request. One approach to hide the identities is by using peer-to-peer relay networks backed by enclaves [33]. Also, it is a requirement to only communicate with the cloud storage using encrypted connections. Even if the file data is encrypted, the metadata can leak group information to every entity listening to the network traffic.

We modeled the cloud storage component using Minio [34], a distributed object store that is fully compatible with the application programming interfaces (APIs) of Amazon S3. As we need to perform operations against the cloud storage from within an enclave, we ported the Java version of the Minio client library to C++ so that it can run together with the WriterShield. These modifications amount to 4000 LoCs of C++, which we openly release[1]. Without accounting for external libraries, the WriterShield consists of 800 LoCs.

3) Client: As part of our prototype implementation, we developed a full-featured client in 1200 LoCs of Kotlin. The client can be set up to operate in all possible configurations of A-Sky: keys in linear or indexed envelopes, *writes* through the WriterShield, or through a standard proxy onto a Minio or Amazon S3 storage back-end with short-lived token-based authentication. Kotlin's full interoperability with the Java ecosystem allows us to easily integrate with the Java Microbenchmark Harness (JMH) [35] and Yahoo! cloud serving benchmark (YCSB) [14] frameworks that we use to perform the evaluation of A-Sky (§VI).

4) Deployment: All our components can be independently replicated to provide availability, fault tolerance or cope with the load. Therefore, we have packaged our micro-services as individual containers, which we then orchestrate using an SGX-aware adaptation of Kubernetes [36]. The proposed deployment considers that there exists a fast data link between the organization premises and the infrastructure where the TEE-enabled micro-services are hosted. As such, our deployment could further benefit from *edge* computing gateways sitting at the border of the organization. Moreover, by considering that attested TEE micro-services are self-contained with respect to the hosting environment, other deployment options arise by elastically handling AccessControl and WriterShield instances between the organization edge and the user or cloud premises, if the latter two are equipped with such capabilities. We leave these deployment options to future work.

VI. Evaluation

We evaluate the performance and scalability of our solution by first conducting micro-benchmarks. Then, we use the well-known YCSB [14] test suite to evaluate the overall system performance.

All our experiments run on a cluster of 5 SGX-enabled Dell PowerEdge R330 servers, each having an Intel Xeon E3-1270 v6 processor and $64\,\mathrm{GiB}$ of memory. Additionally,

[1] https://github.com/rafaelppires/anonym-sharing

TABLE II: Throughput comparison (*i.e.*, group size per second: $|\mathcal{G}|/s$) of A-SKY cryptographic scheme and *BBW* [8], isolating enveloping (Env.) and de-enveloping (Dnv.) operations, in the standard and efficient decryption (ED) mode.

	Env. $[\|\mathcal{G}\|/s]$	Dnv. $[\|\mathcal{G}\|/s]$	Env.^{ED} $[\|\mathcal{G}\|/s]$	Dnv.^{ED} [µs]
BBW	3.3×10^2	$5\ \times10^3$	$3\ \times10^2$	<4
A-SKY	1.9×10^6	2.5×10^6	1.2×10^6	<4
Faster by	3.7 OoM	2.6 OoM	3.6 OoM	*n/a*

we use 3 Dell PowerEdge R630 dual-socket servers, each equipped with 2 Intel Xeon E5-2683 v4 CPUs and 128 GiB of RAM. One of the latter machines is split in 3 virtual machines to handle the roles of Kubernetes master, Minio server and benchmarking client (when a second client is needed). SGX machines use the latest available microcode revision `0x8e`, and have the Hyper-threading feature disabled to mitigate the Foreshadow security flaw [37]. Communication between machines is handled by a Gigabit Ethernet network.

When error bars are shown, they represent the 95 % confidence interval.

1) Micro-benchmarks: **Cryptographic Scheme Performance.** We start the performance evaluation of A-SKY by isolating and measuring the performance of the underlying cryptographic primitive. We employ the ANOBE scheme defined by Barth *et al.* [8] (*BBW*) as a baseline. Our implementation of *BBW* uses an *elliptic curve integrated encryption scheme* as the IND-CCA2 public key cryptosystem used by the original scheme. Both cryptographic schemes key materials (*i.e.*, keys, curve) are chosen to meet 256 bit of *equivalent security strength* [38]. Moreover, we implement the efficient decryption of *BBW*, as suggested in the paper by relying on the hardness of the DH problem, however in the context of much faster *elliptic curves* (ECDH). As the content encryption is similarly implemented for the two schemes, we choose to only measure and present the key enveloping and de-enveloping performance. We consider that the user keys are available at the time of the calls.

Table II shows the speed of cryptographic key enveloping and de-enveloping by reporting the number of group members handled per second. If *BBW* can envelope groups of only 330 members per second, A-SKY can handle 3.7 orders of magnitude (OoM) more users per second. The considerable speed difference is justified by the performance gap between public key (used by *BBW*) and symmetric encryption (used by A-SKY) primitives. Likewise, a performance increase of 2.6 OoMs is observed for the de-enveloping operation. *BBW* provides an *efficient decryption* mode that can achieve fast decryption times (less than 4 µs for the highest tested group size), but with a high cost of only 300 group size envelopings per second. A-SKY is able to support the same efficient decryption speed, by performing 1.2 million group size envelopings per second, a gain of 3.6 OoMs compared to *BBW*. Furthermore, as explained in §V, A-SKY produces a ciphertext of 60 B and 88 B respectively for the standard and efficient decryption modes, per each group member, compared to 126 B and 154 B bytes per member for *BBW*.

Scalability. We further evaluate the throughput of operations performed by administrators when varying the number of ACCESSCONTROL instances. Requests are distributed among

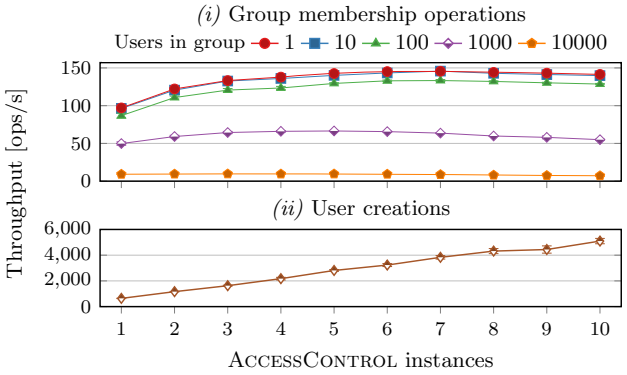

Fig. 3: Throughput achieved by ACCESSCONTROL: *(i)* adding or revoking users to/from groups of various sizes, and *(ii)* creating users.

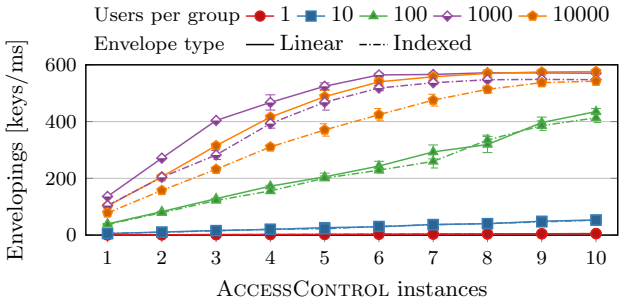

Fig. 4: Throughput of enveloping a message for groups of various sizes with varying instances of the ACCESSCONTROL micro-service.

the instances of ACCESSCONTROL by exposing a *service* in Kubernetes. Fig. 3 shows our results. The scalability of adding a user to a group or revoking its rights is limited, as these operations require to perform one a read-modify-write (RMW) cycle to check and update the signature of the group *document*. The larger the group, the more the operation takes time as each signature encompasses every user within the group. This effect could be mitigated by, *e.g.*, batching multiple operations on a given group together. On the other hand, the operation that creates users scales linearly with the number of ACCESSCONTROL instances, allowing more than 5000 user creations per second with 10 instances.

Next, we evaluated the number of keys that can be included in an envelope per unit of time, also when varying the number of instances of the ACCESSCONTROL service. A close-to-linear trend can be observed according to number of instances in Fig. 4, showing that this operation also benefits from horizontal scalability. With groups of 1000 to 10 000 members, the throughput ceases to increase with more than 7 instances as the MongoDB backend becomes a bottleneck. For smaller groups, the performance is diminished due to the overhead associated with each request (*e.g.*, network connection, REST request, enclave transitions, *etc.*), although increasing the number of ACCESSCONTROL instances provides greater benefits. Additionally, we ran the same experiment with the indexing feature turned on. For groups of 10 000 users, the throughput is reduced by 6 % to 26 %, having a marginal impact on smaller groups where the performance mostly depends on fixed costs.

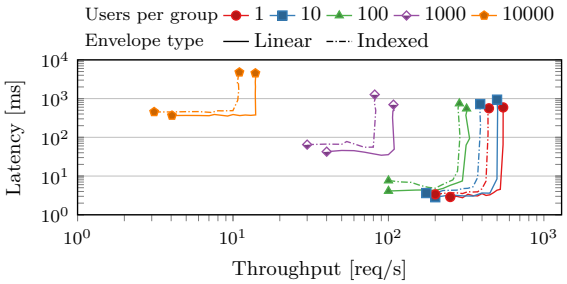

Fig. 5: Throughput *vs.* latency plot of enveloping a message for groups of various sizes.

We also evaluated the latency of the enveloping operation by increasing the throughput until saturation, again with indexing turned off and on. Looking at Fig. 5, we notice that for groups which are larger than 100 users, latency increases linearly according to the group size, while the saturation throughput decreases linearly.

To evaluate the performance of the WRITERSHIELD, we conduct two experiments. In the first one, data written to the cloud is proxied through the TEE. In the second one, the WRITERSHIELD is only used as a facilitator to obtain temporary access tokens for the cloud storage, with write operations being proxied through an NGINX server in TCP reverse-proxy mode. In order to establish a baseline, we also wrote the data directly to the cloud storage service, without any intermediary. Results are shown in Fig. 6. Looking at the bar plot on the left-hand side, we notice that, for files of 1 kB and 10 kB the difference in performance is negligible, whereas bigger files cause more performance degradation when using the token feature. When the WRITERSHIELD is used to forward data instead (right-hand side), we see that the throughput increases with the number of service instances until it seems to plateau at about the same values as with the tokenized variant. For files of 1 MB, adding WRITERSHIELD instances shows no benefit. This effect happens due to the saturation of enclave resources acting as a TLS bridge between clients and the cloud storage server. Overall, using tokens would be the most efficient approach, although in this case the client would be responsible for using adequate proxies in order to hide its identity from the cloud storage.

2) Macro-benchmarks: We use YCSB [14] to observe the behavior of A-SKY under different usage conditions that are specific to data serving systems. We implemented an interface layer to link the benchmarking tool to A-SKY. As our system is not capable of direct-access writes, *update* operations are replaced by RMW operations. In order to capture usage conditions, we run YCSB workloads *A* (update heavy), *B* (read heavy) and *C* (read only), to which we add an *insert*-only workload. We consider files of 3 different sizes from 1 KiB to 1 MiB. We simulate 100 000 operations across 64 concurrent users and report upon the user operation throughput. At times, we add a second simultaneously-running instance of YCSB that simulates 8 administrators doing group membership operations. The administrative operations are equally distributed between adding a user to a group and revoking one, so that the size of the user database stays more-or-less constant.

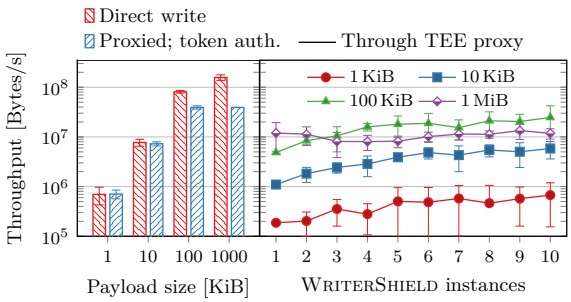

Fig. 6: Throughput of writing data to the cloud storage in different ways: directly (baseline), through a TCP proxy using a temporary token for authentication, and through varying number of in-enclave WRITERSHIELD instances.

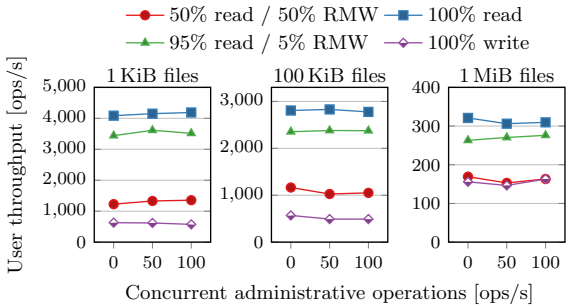

Fig. 7: User throughput observed by our YCSB-based macro-benchmark, with various file access patterns, varying file sizes, and addition of simultaneous administrative operations.

Fig. 7 shows the results of our experiment. One can notice that the user throughput is not influenced by concurrent administrative operations, as each type of operation involves separate components of our architecture. For small files of 1 KiB, an increasing proportion of writes causes a degradation in performance from 4100 ops/s for read only to 628 ops/s for write-only workloads. With larger 1 MiB files, the difference is more nuanced, with a throughput of 320 ops/s for the read-only workload compared to 155 ops/s for the write-only workload. Therefore, the fixed costs are largely dominant when writing small files (*e.g.*, enveloping the file key), but are increasingly amortized for larger file sizes. We can also observe that the throughput in B/s (*i.e.*, multiplying the result in ops/s to the file size) is largely superior for larger files, as we have already noticed in Fig. 6. In a nutshell, we retain that the end-user experience offered by A-SKY is not influenced by concurrent administrative operations, and that the overhead of the additional operations required for writing become smaller for larger files.

VII. CONCLUSION

We introduced A-SKY, an end-to-end system that guarantees anonymity and confidentiality of shared content (*e.g.*, files). A-SKY leverages trusted execution environments (TEEs) exclusively for the content sharing operation, while TEE capabilities are not required for end users consuming the shared content.

We have introduced a novel anonymous broadcast encryption (ANOBE) scheme that exploits additional assumptions about the availability of a TEE compared to state-of-the-art schemes, in order to achieve fast and practical performance for its operations. We incorporated the novel cryptographic construction into a scalable system design that leverages micro-services to elastically scale per the undergoing access control and data sharing workloads. Results indicate that our cryptographic scheme is faster than the state-of-the-art ANOBE schemes by 3 orders of magnitude. An end-to-end system that utilizes our scheme can serve groups of 10 000 users with a throughput of 100 000 key derivations per second per service instance.

ACKNOWLEDGMENT

The research leading to these results has received funding from the French Directorate General of Armaments (DGA) under contract RAPID-172906010.

REFERENCES

[1] A. Bessani, R. Mendes, T. Oliveira, N. Neves, M. Correia, M. Pasin, and P. Verissimo, "SCFS: A shared cloud-backed file system," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 169–180.

[2] S. J. Dwyer III, A. C. Weaver, and K. K. Hughes, "Health insurance portability and accountability act," *Security Issues in the Digital Medical Enterprise*, vol. 72, no. 2, pp. 9–18, 2004.

[3] iDeals Virtual Data Rooms. (2019). Share and collaborate on business-critical documents in a secure way, [Online]. Available: https://www.idealsvdr.com/.

[4] 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 *Security and Privacy (SP), 2016 IEEE Symposium on*, IEEE, 2016, pp. 819–838.

[5] 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.

[6] S. Contiu, R. Pires, S. Vaucher, M. Pasin, P. Felber, and L. Réveillère, "IBBE-SGX: Cryptographic group access control using trusted execution environments," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 207–218.

[7] S. Angel and S. Setty, "Unobservable communication over fully untrusted infrastructure," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 551–569.

[8] A. Barth, D. Boneh, and B. Waters, "Privacy in encrypted content distribution using private broadcast encryption," in *International Conference on Financial Cryptography and Data Security*, Springer, 2006, pp. 52–64.

[9] B. Libert, K. G. Paterson, and E. A. Quaglia, "Anonymous broadcast encryption: Adaptive security and efficient constructions in the standard model," in *International Workshop on Public Key Cryptography*, Springer, 2012, pp. 206–224.

[10] V. Costan and S. Devadas, "Intel SGX explained.," *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[11] T. Alves and D Felton, "TrustZone: Integrated hardware and software security," *White paper*, 2004.

[12] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "VC3: Trustworthy data analytics in the cloud using SGX," in *Security and Privacy (SP), 2015 IEEE Symposium on*, IEEE, 2015, pp. 38–54.

[13] S. Brenner, C. Wulf, D. Goltzsche, N. Weichbrodt, M. Lorenz, C. Fetzer, P. Pietzuch, and R. Kapitza, "SecureKeeper: Confidential ZooKeeper using Intel SGX," in *Proceedings of the 17th International Middleware Conference*, ACM, 2016, p. 14.

[14] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing*, ACM, 2010, pp. 143–154.

[15] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[16] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, "DepSky: Dependable and secure storage in a cloud-of-clouds," *ACM Transactions on Storage (TOS)*, vol. 9, no. 4, p. 12, 2013.

[17] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[18] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Annual International Cryptology Conference*, Springer, 2005, pp. 258–275.

[19] J. Li, C. Qin, P. P. Lee, and J. Li, "Rekeying for encrypted deduplication storage," in *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*, IEEE, 2016, pp. 618–629.

[20] 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.

[21] P. Rsler, C. Mainka, and J. Schwenk, "More is less: On the end-to-end security of group chats in Signal, WhatsApp, and Threema," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018, pp. 415–429.

[22] P. R. Zimmermann, *The official PGP user's guide*. MIT press, 1995.

[23] D. Boneh, E. Shen, and B. Waters, "Strongly unforgeable signatures based on computational Diffie-Hellman," in *International Workshop on Public Key Cryptography*, Springer, 2006, pp. 229–240.

[24] P. MacKenzie, M. K. Reiter, and K. Yang, "Alternatives to non-malleability: Definitions, constructions, and applications," in *Theory of Cryptography Conference*, Springer, 2004, pp. 171–190.

[25] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 619–636.

[26] F. Shaon, M. Kantarcioglu, Z. Lin, and L. Khan, "SGX-BigMatrix: A practical encrypted data analytic framework with trusted processors," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2017, pp. 1211–1228.

[27] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure linux containers with intel SGX," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 689–703.

[28] M. Backes, C. Cachin, and A. Oprea, "Secure key-updating for lazy revocation," in *European Symposium on Research in Computer Security*, Springer, 2006, pp. 327–346.

[29] MongoDB Inc. (2019). MongoDB, [Online]. Available: https://www.mongodb.com/.

[30] ——, (2018). MongoDB C driver. version 1.12.0, [Online]. Available: http://mongoc.org/libmongoc/1.12.0/index.html.

[31] N. Lohmann *et al.* (2018). JSON for modern C++, [Online]. Available: https://github.com/nlohmann/json.

[32] J. Han. (2017). OpenSSL library for SGX application, [Online]. Available: https://github.com/sparkly9399/SGX-OpenSSL.

[33] R. Pires, D. Goltzsche, S. B. Mokhtar, S. Bouchenak, A. Boutet, P. Felber, R. Kapitza, M. Pasin, and V. Schiavoni, "CYCLOSA: Decentralizing private web search through SGX-based browser extensions," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 467–477.

[34] Minio, Inc. (2019). Minio: Private cloud storage, [Online]. Available: https://www.minio.io/.

[35] A. Shipilev *et al.* (2018). JMH: Java microbenchmark harness, Oracle Corporation, [Online]. Available: https://openjdk.java.net/projects/code-tools/jmh/.

[36] S. Vaucher, R. Pires, P. Felber, M. Pasin, V. Schiavoni, and C. Fetzer, "SGX-aware container orchestration for heterogeneous clusters," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 730–741.

[37] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 991–1008.

[38] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "NIST special publication 800-57," *NIST Special publication*, vol. 800, no. 57, pp. 1–142, 2007.

[39] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2000, pp. 531–545.

[40] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov, "Iron: Functional encryption using Intel SGX," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2017, pp. 765–782.

APPENDIX

Security analysis. We discuss the security guarantees of A-SKY and provide a brief intuition for the formalism of a reductionist security proof. We hypothesize that A-SKY achieves indistinguishability with respect to adaptively-chosen ciphertexts (*i.e.*, IND-CCA2) according to our targeted threat model (§II). Within IND-CCA2 security, the adversary can be an active member of the group and therefore can rightfully decrypt group messages. Such an attacker is allowed to try as many additional group encryptions (*i.e.*, key envelopings) of arbitrarily constructed groups, without being able to infer if the resulted ciphertexts (*i.e.*, *envelopes*) are pointing to the same group members. Note that proving A-SKY as IND-CCA2 implicitly assures security guarantees against *non* adaptive chosen ciphertext (IND-CCA) and plaintext (IND-CPA) attacks. Differently than IND-CCA2, IND-CCA assumes that the adversary is given only one chance to try a set of group encryptions. Within IND-CPA, also known as *semantic security*, the adversary is a passive group member that only observes and does not have the ciphertext choice capability. Intuitively, the security guarantees extend to adversaries that are not members of a group.

Before laying out the security proof sketch, we recall the two pillars of A-SKY: authenticated encryption (AE) and trusted execution environments (TEEs). AE primitives are considered secure in the adaptive chosen ciphertext attack when employing the *encrypt-then-mac* composition method [39]. Such a guarantee forces to choose a specific AE mode for advanced encryption standard (AES), as described in §V. On the other side, TEEs have been used in the composition of functional encryption cryptographic primitives shown to achieve IND-CCA2 guarantees [40]. In the following, we retain the formalism of Fisch *et al*. [40] that abstracts TEEs as a *secure hardware scheme*.

Theorem 1. Assuming that AE is IND-CCA2 and a TEE is a *secure hardware scheme*, then A-SKY is IND-CCA2.

Proof (intuition). We provide the sketch of a reductionist method that lays the frame for a formal proof. A-SKY can be seen as a reduction of the anonymous broadcast encryption scheme of Barth *et al*. [8] (*BBW*), by considering two arguments: (1) AE in conjunction with a *secure hardware scheme* replaces the public key encryption (PKE) scheme, and (2) *secure hardware scheme* signatures replace the strongly unforgeable signature. As *BBW* has been proved IND-CCA2 secure by Libert *et al*. (Theorem 1 of [9]), relying on the two aforementioned replacements, one can construct identical adversary-challenger game steps (Def. 2 in [9]) and employ a similar sequence of experiments (Appendix A in [9]) that can prove that A-SKY is immune to chosen ciphertext attacks. As such, in the formal language of computational security proofs, A-SKY security relies on the assumption that an attacker would be unable to employ a polynomial time Turing machine for breaking the computational hardness of authenticated encryption and the robustness of TEEs.