Applied Cryptographic Access Control for Untrusted Cloud Storage

Stefan Contiu

ACADÉMIE DE BORDEAUX
UNIVERSITÉ DE **B**ORDEAUX
Sciences et **T**echnologies

THÈSE

Présentée au Laboratoire Bordelais de Recherche en Informatique pour obtenir le grade de Docteur de l'Université de Bordeaux

Spécialité	:	**Informatique**
Formation Doctorale	:	**Informatique**
École Doctorale	:	**Mathématiques et Informatique**

Applied Cryptographic Access Control for Untrusted Cloud Storage

par

Stefan CONTIU

Soutenue le 13 Novembre 2019, devant le jury composé de :

Président du jury
Guy MELANÇON, Professeur ..... Université de Bordeaux, France

Directeur de thèse
Laurent RÉVEILLÈRE, Professeur ..... Université de Bordeaux, France

Rapporteurs
Romain ROUVOY, Professeur ..... Université de Lille, France
Rüdiger KAPITZA, Professeur ..... Technische Universität Braunschweig, Allemagne

Examinateurs
Etienne RIVIÈRE, Professeur ..... UCLouvain, Belgique
Sonia BEN MOKHTAR, Directrice de Recherches ..... CNRS, France
Paulo ESTEVES-VERÍSSIMO, Professeur ..... Université du Luxembourg, Luxembourg

Membres invités
Thierry LEBLOND, President ..... Scille, France
Emmanuel LEBLOND, Directeur Technique ..... Scille, France
Philippe CLERMONT, Secrétaire Général et Innovation ..... Scille, France

Abstract

Public clouds enable *storing* and *sharing* data with efficient cost and high availability. Nevertheless the benefits, cloud providers are recurrently breached by malicious users exposing sensitive user content. To mitigate the lack of security guarantees, users can impose *end-to-end* security by encrypting the data before remotely storing it.

Access control mechanisms specify the users who are allowed to produce or consume the remote data. As data is encrypted, access control is performed cryptogrpahically, concealed from the cloud storage. Cryptographic key management is used for regulating user access while re-encryption techniques are used for key updates. State-of-the-art key management often trades computational time for storage footprint, while re-encryption techniques exchange great security guarantees for speed. In the context of very large and highly dynamic cloud specific workloads, state-of-the-art cryptographic access control is generally inefficient.

This thesis proposes a minimal integration of Trusted Execution Environments (TEE) to achieve efficient access control. Leveraging TEE, we perform a change in assumptions of traditional key distribution schemes, deriving a *confidential* and an *anonymous* scheme, both achieving efficient computational latency and low storage footprint. In addition, we propose a lightweight data re-encryption method by processing only portions of the data in TEE directly at the provider side. We carry out a comprehensive implementation and evaluation using Intel Software Guard Extensions (SGX) as TEE. Benchmarking results highlight that our key management and re-encryption schemes are few orders of magnitude better than state-of-the-art.

Keywords: *security; privacy; cloud; storage; access control; confidentiality; anonymity.*

Résumé

Les clouds publics permettent de *stocker* et de *partager* des données à faible coût et haute disponibilité. Néanmoins, les avantages, les fournisseurs de cloud sont contournés de manière récurrente par des utilisateurs malveillants exposant des contenus utilisateurs sensibles. Face au manque de garanties de sécurité, les utilisateurs peuvent imposer une sécurité de *bout-en-bout* en chiffrant les données avant de les stocker à distance.

Les mécanismes de contrôle d'accès filtrent les utilisateurs autorisés à produire ou à utiliser les données distantes. Au fur et à mesure que les données sont chiffrées, le contrôle d'accès est effectué de manière cryptographique, indépendamment du stockage en nuage. La gestion des clés cryptographiques régule l'accès des utilisateurs, tandis que des techniques de rechiffrement sont utilisées pour les mises à jour de clés. Une gestion des clés permet souvent d'arbitrer entre le temps de calcul et l'empreinte de stockage, tandis que les techniques de rechiffrement arbitrent entre les garanties de sécurité et la rapidité. Dans le cas de très volumineuses et dynamiques charges de travail spécifiques sur le cloud, un contrôle d'accès cryptographique même performant est généralement inefficace.

Cette thèse propose une intégration minimale des environnements d'exécution de confiance (TEE) pour obtenir un contrôle d'accès efficace. En utilisant TEE, nous modifions les hypothèses des schémas de distribution de clés traditionnels, en dérivant deux schémas, un *confidentiel* et un *anonyme*, permettant à la fois d'obtenir une latence informatique supportable en même temps qu'une faible empreinte de stockage. De plus, nous proposons une méthode légère de rechiffrement des données en ne traitant que des parties des données dans TEE directement chez le fournisseur. Nous réalisons une mise en œuvre et une évaluation complètes en utilisant Intel Software Guard Extensions (SGX) comme TEE. Les résultats de l'analyse comparative montrent que nos systèmes de gestion de clés et de rechiffrement accroissent l'état de la technique de plusieurs ordres de grandeur.

Mots clés : *sécurité; intimité; cloud; stockage; contrôle d'accès; confidentialité; l'anonymat.*

LaBRI — 351, Cours de la Libération — 33400 Talence — France

Dedicated to Ioanna,
Je remercie Dieu pour chaque erreur que j'ai faite, parce que chacune d'elles
m'a indiqué le chemin qui m'a mené à toi.

Acknowledgment

The author would like to thank the thesis director *Laurent Réveillère* for his dedicated and passionate guidance, and to *Thierry Leblond*, *Philippe Clermont* and *Emmanuel Leblond* from Scille for their encouragement, trust and appreciation vividly shown throughout the journey. The findings of this thesis would not have been possible without the scientific collaborators : *Rafael Pires*, *Sébastien Vaucher*, *Pascal Felber*, and *Marcelo Pasin* from University of Neuchâtel (Switzerland); *Etienne Rivière* from UCLouvain (Belgium); *Adrien Luxey* and *David Bromberg* from University of Rennes (France); *Simon Da Silva* and *Daniel Négru* from LaBRI (France); *Sonia Ben Mokhtar* from CNRS Lyon (France).

Contents

CHAPTER

1 Introduction

This thesis was conducted in an industrial partnership between *Scille* and *LaBRI*. In these lines, the research activity focused exclusively on end-to-end secure cloud storage. First, we identify unanswered research challenges from *state-of-the-art* literature. Second, we address these challenges by efficient novel constructions. This chapter introduces the thesis context, cloud storage, end-to-end security and access control, and a tour of our efficient security contributions.

1.1 Industrial Context

The presented work was conducted in partnership with *Scille*[1]. *Scille* is a French software development company founded in 2016, servicing notable clients such as the French *Ministry of Interior* or *Ministry of Defense*. *Scille* also performs a rich research activity. Within this context, *Scille* received financing from the *Ministry of Defense* through a DGA project[2] between Feb. 2017 to Jun. 2019 to develop PARSEC[3], an open source end-to-end secured file storage and sharing solution.

The main driver of the research activity has been identifying large scale design challenges ahead of engineering work. At the time of writing (Oct. 2019) PARSEC v.1 *Community Edition* is in beta-testing. The subsequent version PARSEC v.2 *Enterprise Edition* integrates entirely the research results presented within this dissertation. PARSEC v.2 *Enterprise Edition* targets seamless integration with large size organizations, accommodating a significant operational workload and data volumetry.

1. https://www.scille.fr/
2. Direction Générale de l'Armement (DGA) RAPID 172906010
3. https://parsec.cloud/

1.2 Cloud Storage Today

Cloud storage receives much attention recently and shows a fast adoption by individuals, businesses or governmental institutions. Not surprisingly, various start-ups emerged to fill the demand (e.g., Dropbox, FileCloud), while technology big-players leaped to cover the same need (e.g., Microsoft's OneDrive, Google Drive).

The popularity of cloud storage among end-users comes from the easiness of *safeguarding* and *sharing* personal data [Seybert and Reinecke, 2014]. If priory such activities were done using direct-attached storage (e.g., hard drives), universal serial bus (USB) flash drives, or emailing, the cloud storage offers a much simpler and cost efficient alternative. Assuming the possession of an internet connection, cloud storage gives to end-users the impression of handling locally stored data. The background task of moving this data *to* and *from* the cloud is abstracted from them. Such services became so popular that Dropbox alone reached more than 500 million active users in 2018 [Forbes, 2018].

The adoption of cloud storage by organized establishments can be further argued by benefits. Priory, organizations typically maintained in-house data storage privately accessible through the organization network. Examples are dedicated storage units such as Network Attached Storage (NAS) or multiple interconnected storage units forming a Storage Area Network (SAN). However, in-house storage has inconveniences. The hosting organization needs time, resources and therefore costs to set-up and maintain hardware and software components of the solution. Moreover, an in-premises solution is rigid, it risks overflowing over the maximum capacity or contrary to be under utilized. Cloud storage mitigates the inconveniences by shifting the responsibilities from the hands of the organization into the hands of the cloud provider. The organization spending cost is lowered, while the storage space can grow or shrink depending on use.

To enrich the landscape of cloud storage services, providers introduced the paradigm of *object*, namely Binary Large Object (BLOB) storage. Examples are Amazon's Simple Storage Service (S3), Google Cloud Storage and Microsoft Azure Storage. The object paradigm is a simplification of the hierarchical file storage model by considering each stored resource as a unique immutable object. This model reduces metadata and achieves better scalability. In this line of service, cloud providers offer additional competitive benefits. Customers can chose to disperse their objects over multiple geographical locations, increasing data reliability and availability. Customers can chose subscription plans depending on the frequency of data access - expressed by thermic quantifiers (*hot* or *cold* data) - or pay only for the consumed resource (space or bandwidth) referred to as *pay-as-you-go*.

Throughout, the adoption of cloud storage by individuals and organizations is closely mirrored by the growth of its business market. In 2016 the cloud storage market reached $24 billion and forecast to grow to $75 billion in 2021 [MarketWatch, 2016].

1.3 Securing Cloud Storage

Nevertheless the benefits, security concerns hover the use of cloud storage. News repeatedly emerge of service providers breached by malicious attackers resulting in sensitive user data exposed. For example, in 2016, it was found that 68 million Dropbox accounts have been potentially compromised in an incident four years before [Guardian, 2016]. In another case, in 2016 Apple iCloud storage was breached exposing private photos of celebrities [Tripwire, 2016]. In 2018 it was revealed that an Amazon employee potentially sold confidential user data to unauthorized parties [CNBC, 2018]. In 2018, the United States enacted the Cloud-Act, allowing American authorities to warrant on cloud providers data even when stored on foreign soil, attracting controversies over international sovereignty [Numerama, 2018].

Not surprisingly, both individuals and organizations list security as the main concern for adopting cloud backed services [Computing, 2018]. While individuals mostly see the threat endangering privacy as a civil right, business organizations can suffer major financial issues. For example, the notorious data breach of Equifax in 2017 was estimated to cost the company $275 million [Reuters, 2017].

To mitigate the lack of proper security guarantees while enabling their use, one can enforce the *end-to-end security* paradigm when using cloud storage.

End-to-End Security

The end-to-end security model considers that except the communication end-points everyone is an adversary. As discussed, the storage provider can not be considered entirely trusted - not necessarily by arguing its honesty, but by the potential threat of being breached by malicious actors. Therefore, the storage provider is categorized as an adversary. In the same way the communication links between the end-points and the provider are potentially adversarial. Figure 1.1 illustrates the end-to-end security concept, enforcing the users (end-points) as the only trusted parties.

The security requirements enforced by the end-points vary depending on the desired security guarantees. For example *confidentiality* states that only end-users should be capable to comprehend the data. *Integrity* assures that data is not altered while *authenticity*

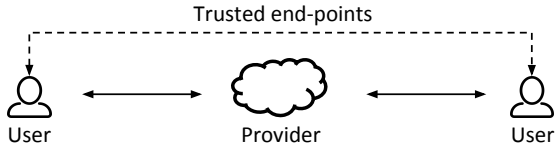


Figure 1.1 – Trusted end-points in end-to-end security.

enables tracking with certainty the data originator. Sometimes, *anonymity* is desired for hiding the identities of communicating parties.

These security properties are usually addressed by cryptographic primitives (a gentle primer on cryptography is included in Section 2.1). Executing the cryptographic primitives on the user side, keeps the key material known only to the end-points and never to the cloud provider. Confidentiality is achieved by *symmetric* encryption, while integrity and authenticity are achieved by *hash* methods and *digital signatures*. Imposing the security properties to a well-defined group of users is achieved by the means of *public key* cryptography.

In the research literature a number of remote storage systems integrate *end-to-end security* through cryptographic primitives [Li et al., 2016a,b; Han et al., 2015; Dobre et al., 2014; Bessani et al., 2014, 2013; Popa et al., 2011a; Wilcox-O'Hearn and Warner, 2008; Kubiatowicz et al., 2000]. Differently, certain systems achieve *end-to-end* confidentiality by assuming the non-collusion of multiple storage providers, thus relying on the security strength of dispersal methods [Tang et al., 2015; Resch and Plank, 2011; Chung et al., 2015].

End-to-end security was applied in other areas of systems research, such as relational databases [Bajaj and Sion, 2013; Poddar et al., 2016; Popa et al., 2011b], email [Koh et al., 2019; Ruoti et al., 2016] or content-based routing [Pires et al., 2016].

In the industrial landscape, even though *end-to-end security* has been integrated in popular instant communication applications (e.g., WhatsUp, Signal [Rösler et al., 2018]), it lacks wide adoption for public cloud storage applications.

Cryptographic Access Control

Access control policies specify the users who can access a certain resource (e.g., file or object). Within *end-to-end* security, the users - i.e., the end-points - have the role of enforcing access control. As such, access control is performed in a cryptographic manner, un-comprehensible to the adversarial cloud provider. Cryptographic access control is enforced by mechanisms that operate over two distinct dimensions. If the first mechanism operates at the level of users (e.g., identities), the second dimension operates at the level of the data (e.g., resource).

Users Dimension. In *end-to-end* security the data shared among multiple users is encrypted with a *secret key* known only by the users. Therefore, specifying and enforcing the users that can access the data is performed by a distribution mechanism of the *secret key*. The simplest method, popularly referred to as Hybrid Encryption (HE), symmetrically encrypts the data with the *secret key*, and then encrypts this *key* with public-key cryptography - such as RSA or Eliptic Curve Cryptography [Goh et al., 2003]. Other approaches rely on Pairing-Based Cryptography (PBC) and offer different levels of granularity. A few examples include : Identity-Based Encryption (IBE) [Boneh and Franklin, 2001] which works similarly to public-key encryption at the identity level; or Identity-Based Broadcast Encryption (IBBE) [Delerablée et al., 2007] that can capture group-like policies. Moreover, certain key

distribution schemes guarantee the secrecy of user identities, such as Anonymous Broadcast Encryption (ANOBE) [Barth et al., 2006].

Data Dimension. During an access control policy change, besides the distribution of a new encryption key by aforementioned methods, the encrypted data needs to assimilate this key change. Clearly, all data created after the policy change will be encrypted by the new key. However, previously created data, needs to be re-encrypted with the new key. Existing approaches are either low-cost by delaying re-encryption until the resource is updated (*lazy revocation* [Kallahalla et al., 2003; Backes et al., 2005]), or prevent any further access to the entire set of data through a high-cost full re-encryption of all resources (*active revocation*).

Efficiency Issues

In the context of cloud storage, cryptographic access control can suffer prohibitive cost. Differently than traditional storage, access control policies for cloud storage are potentially large and highly dynamic [Garrison et al., 2016]. Nowadays, large scale organizations surpass hundreds of thousands of employees [GeekWire, 2018], while companies selling access to remotely stored data surpass millions of clients [CNN, 2019]. Moreover, data vendors utilizing public cloud storage reach data-set sizes in the range of Petabytes [Toman, 2017; Genomics, 2018].

Enforcing cryptographic access control over such high loads, makes both *user* and *data* access control dimensions ineffective. As illustrated by our preliminary benchmarks (Sections 2.4 and 2.5), managing users by *state-of-the-art* key distribution methods is inefficient due to the computational latency (e.g., hours) or large metadata (e.g., hundreds of Megabytes) unsuitable for high frequency access control policy changes. Moreover, using *lazy revocation* by delaying re-encryption to the first update does not apply to *immutable* resources - the content type published by data vendors. Instead, such vendors need to rely on the prohibitive *active revocation* or *skip* the re-encryption process.

Instruments for Efficiency

There exist a tension between the *efficiency* and the *security* of cryptographic access control. However, by agreeing on certain security assumptions, one can balance this trade-off in favor of efficient constructions. We present our efficiency empowering instruments in the form of three security assumptions.

1. Administrators Perform Access Control. This assumption mimics the functioning of large organizations in which only privileged users perform access control operations (e.g., network administrators, IT-desk etc.). Differently, in *state-of-the-art* key distribution schemes [Boneh et al., 2005b; Delerablée et al., 2007] user management is performed by ordinary users.

2. Availability of Trusted Execution Environments (TEE). TEE are processor extensions that can provide shielded code execution while guaranteeing the isolation, confidentiality and integrity of the computations. Intel SGX [Costan and Devadas, 2016] has been successfully used for performing trusted computations in untrusted environments [Schuster et al., 2015; Brenner et al., 2016]. We constrain this availability assumption to the smallest set of system actors. In other words we can not assume that everybody is equipped with Intel SGX capabilities. Moreover, we require that the performance overhead induced by adopting TEE is minimal.

3. Local Caching of Plaintext. Users that entirely downloaded and *viewed* a resource (e.g., file) can not be prevented to locally cache and access this copy even after they are evicted from the access control policy. Therefore, similarly to related work [Li et al., 2016a; Myers and Shull, 2017], we assume that the adversary did not previously downloaded the entirety of the resource.

1.4 Our Contributions

We propose three unique contributions CON-SKY, ANO-SKY and REV-SKY[4]. Each contribution takes shape by joining the aforementioned efficiency instruments to a concrete cryptographic access control problem at large scale. We deduct new theoretical constructs, and shell them within end-to-end systems over cloud storage to prove their efficiency. Our first two contributions tackle the user management dimension of access control, and propose a *confidential* and an *anonymous* key distribution scheme respectively. Our third contribution focuses on the data management dimension of access control, and proposes an efficient *active revocation* method.

CON-SKY: Confidential Access Control using TEE.

Our first contribution CON-SKY is a cryptographic access control extension that is efficient both in terms of computation and storage even when processing large and dynamic workloads of membership operations. CON-SKY leverages the first two instruments of efficiency, thus requiring administrators to perform user management operations in TEE. CON-SKY builds upon Identity-Based Broadcast Encryption (IBBE) [Delerablée, 2007]. IBBE's impracticality for cloud deployments is addressed by exploiting Intel SGX to derive cuts in the computational complexity. Moreover, a partitioning mechanism is proposed such that the computational cost of membership update is bound to a fixed constant partition size rather than the size of the whole user set.

4. The names CON-SKY, ANO-SKY, REV-SKY were inspired by the seminal work of DEP-SKY [Bessani et al., 2013].

Results highlight that CON-SKY performs membership changes 1.2 orders of magnitude faster than the traditional approach of Hybrid Encryption (HE), producing metadata that are 3 orders of magnitude smaller than HE.

ANO-SKY: Anonymous Access Control using TEE.

Our second contribution is ANO-SKY, a cryptographic access control extension capable of providing not only confidentiality but also anonymity guarantees, all while efficiently scaling to large organizations. Similarly to the first contribution, ANO-SKY leverages TEE such as Intel SGX to address the impracticality of Anonymous Broadcast Encryption (ANOBE) [Barth et al., 2006], achieving faster execution times and shorter ciphertexts.

Differently than CON-SKY, ANO-SKY proposes a scalable design leveraging microservices that preserves strong security guarantees while being able to efficiently manage realistic large user bases.

Results highlight that the ANO-SKY cryptographic scheme is 3 orders of magnitude better than state of the art ANOBE, and an end-to-end system encapsulating ANO-SKY can elastically scale to support groups of 10,000 users while maintaining processing costs below 1 second.

REV-SKY: Efficient Revocation using TEE.

Our third contribution is REV-SKY, a practical *active revocation* mechanism that guarantees the same level of protection as full re-encryption, for a fraction of its cost. Our scheme leverages TEE as an efficiency instrument in conjunction with the assumption that adversaries did not entirely downloaded and *viewed* the data resource. Moreover, we utilize an *all-or-nothing* [Rivest, 1997] transformation together with Intel SGX to re-encrypt only small portions of the content directly in the cloud. Distributed execution enables REV-SKY to scale-out over a cluster of re-encryption workers.

Evaluation in a realistic environment shows that REV-SKY outperforms *active revocation* on complete files by up to 3 orders of magnitude on industry workloads.

Remark.

Even tough the thesis main motivation is to securely share data in a cloud environment, the proposed constructs can be adapted to securely broadcast arbitrary information to a group of users over any shared media. Some other examples are peer-to-peer networks or pay-per-view TV.

1.5 Thesis Outline

This dissertation document is structured as follows. After this introductory Chapter, we detail the building blocks and the considered model in Chapter 2. Chapter 3 discusses related work from both security systems and applied cryptography perspectives. We largely discuss our first contribution CON-SKY in Chapter 4, our second contribution ANO-SKY in Chpater 5 and our third contribution REV-SKY in Chapter 6. Within Chapter 7 we present a performance analysis of our three contributions covering micro-benchmarks and realistic use cases. Finally, we conclude the document in Chapter 8 providing concluding remarks and discuss future extensions.

CHAPTER

2 Background

This chapter details the building blocks for constructing end-to-end secure storage systems enabled by cryptographic access control. We lay out a concrete model that details the actors and the necessary assumptions. Next, we provide a quick primer of basic cryptographic primitives, the foundation of cryptographic protocols. We detail our main instrument of efficiency Trusted Execution Environments (TEE) and dive into the details of Intel Software Guard Extensions (SGX). We then investigate existing methods for confidential and anonymous data sharing. Moreover, we investigate efficient re-encryption techniques leveraging All-Or-Nothing transformation. We present the limitations of surveyed methods and provide an overview of open challenges.

2.1 A Primer on Cryptography

To construct cryptographic enabled systems one needs to comprehend the basic building blocks of cryptographic primitives. Cryptography versed readers can skip this subchapter. We detail the functioning of symmetric encryption, message authentication codes, hash messages, public key encryption and digital signatures. For a thorough overview the reader is referred to seminal textbooks [Stinson, 2005; Ferguson and Schneier, 2003].

Symmetric Encryption.

A symmetric-key algorithm provides data confidentiality by the use of the same secret key for both encrypting and decrypting the data. We denote by $E_k(p) \rightarrow c$ a symmetric encryption primitive employing key k on the plaintext p and producing ciphertext c. $D_k(c) \rightarrow p$ is the corresponding decryption operation.

Among existing algorithms, the one defined by the AES specification has become the de facto standard and is used worldwide [Chown, 2002]. It is a block cipher algorithm, operating on fixed-length group of 128 bits called a *block* with a key size of 128, 192 or 256 bits. To securely transform amounts of data larger than a block, the cipher's single-block operation needs to be repeatedly applied accordingly to a block cipher mode. Many modes of operation have been defined [Ferguson and Schneier, 2003], each one offering a different level of performance and robustness.

Message Digests.

Message digests or simply hash functions are one-way collision resistant functions, mapping an input data block to a short fixed size output. The role of hash functions is to provide integrity guarantees over the data. Also, they are utilized as a preceding operation in digital signature schemes, reducing an arbitrarily large amount of data to a small output on which the signature is applied. We denote a message digest simply by $h(d)$.

Hash functions work by splitting the data into fixed size blocks, and iteratively applying a compression function with an intermediate state [Ferguson and Schneier, 2003]. Secure Hash Algorithms (SHA) are a class of secure hashes standardized by NIST in three family sets (SHA-1, SHA-2, and SHA-3). The first set has been proved insecure due to collision attacks – two inputs producing the same hash [Wang et al., 2005]. The second set is a popular choice producing outputs of 256, 384, and 512 bits. Lastly, the third family SHA-3 was standardized by NIST – not as a replacement to SHA-2 but as a robust alternative if someday the latter is to be found insecure.

Authenticated Encryption by Message Authentication Codes (MAC).

Message Authentication Codes (MAC) can prove the integrity and authenticity of a symmetrically encrypted message with respect to the encryption key. As such, the MAC can be used to detect any change to a ciphertext that have transited an untrusted party. Symmetric encryption modes that produce a MAC are called *authenticated* encryption modes. We denote by $AE_k(p) \rightarrow (c,t)$ an authenticated encryption primitive that besides the ciphertext c produces the MAC t, and by $AD_k(c,t) \rightarrow \{p, \perp\}$ the authenticated decryption that either successfully decrypts the plaintext p or fails if the MAC is not valid.

AES Galois Counter Mode (GCM) is a block cipher mode that performs both encryption and authentication by using operations in a finite (Galois) field. GCM is defined for block ciphers with a block size of 128 bits. Implementing GCM can make efficient use of Carry-less Multiplication (CLMUL), an extension to the x86 instruction set used by microprocessors from Intel and AMD [Gueron and Kounavis, 2010].

Public Key Encryption.

Differently than symmetric encryption, public key encryption assumes that each user has two keys, one *public* and one *private* kept secret by the user. This asymmetry enables one party to encrypt a message for a different party without the two possessing a shared secret. As such, the encryption operation (denoted by $E_{PK}(p) \rightarrow c$) uses the public key PK of the destinatary while the decryption (denoted by $D_{pri}(c) \rightarrow p$) is done by the recipient using the private key (pri). The most popular implementations of public key encryption rely on RSA or Elliptic Curve Cryptography.

RSA is a public key cryptosystem based on the difficult mathematical problem of factoring the product of two arbitrarily large prime numbers. The key sizes employed by RSA require a much larger length as compared to symmetric encryption, because solving the mathematical problem is faster than a brute force attack iterating over all possible keys.

Elliptic Curve Cryptography (ECC) is a relatively novel direction in public key cryptosystems [Koblitz, 1987], that besides a considerable interest from academia has also been integrated within technical solutions like Bitcoin, SSH, and TLS [Bos et al., 2014]. The advantage of ECC over the traditional RSA is the small nature of key sizes, implying an increase of computational speed. ECC is based on the difficult mathematical problem of discrete logarithm when the computations are performed over the points of an elliptic curve. The security of the ECC cryptosystem is highly correlated to the choice of the curve equation. Various curves have been proposed and formally reviewed, such as the ones standardized by NIST.

Digital Signatures.

Digital signature algorithms are employed for proving the authenticity of a data block with respect to the user private key. Moreover, they provide the properties of non-repudiation and integrity, meaning that the signing users can not deny themselves as signers and that the data block content is not altered. The verification of the message and signature pair can be openly performed by anybody knowing the user public key. We denote by $S_{pri}(p) \rightarrow \sigma$ and $V_{pub}(p,\sigma) \rightarrow \{true, \perp\}$ the signature and verification primitives.

The popular implementations of digital signatures – similarly to public key encryption – rely on the aforementioned RSA or Eliptic Curve Cryptography.

Security Guarantees.

Semantic Security. A cryptographic scheme is considered secure if it achieves *semantic security*. In a simplistic manner, *semantic security* guarantees that attackers who are given a set of plaintexts and a set of ciphertexts, can not deduce which ciphertext corresponds to which plaintext unless they have knowledge of the key. As the attackers can chose the plaintexts at will, *semantic security* is also referred to as *Indistinguishability under Chosen Plaintext Attack*, or shortly IND-CPA.

Chosen Ciphertext Security. When the attackers are given the option of choosing the set of ciphertexts at will, and also given the decryption of chosen ciphertexts, if they are unable to deduce the secret key then we say that the scheme guarantees *Indistinguishability under Chosen Ciphertext Attack*, or shortly IND-CCA.

A Note on Performance.

When used in the context of securing remote storage, and not necessarily for access control, the implementation choice of cryptographic primitives influences the system performance. As such, the choice of a specific configuration highly depends on the expected security level, the size and type of data to store and the access pattern to these data. The best scheme for a given situation, such as a write-heavy workload of mostly small files, is not necessarily the most appropriate for a different situation such as a read-only workload of large files. For concrete details over the *pros* and *cons* of properly choosing cryptographic primitives the reader is referred to our practical experience report [Contiu et al., 2017].

2.2 Trusted Execution Environments

We propose relying on Trusted Execution Environemtns (TEE) and namely Intel Software Guard Extensions (SGX) as an instrument for efficient security. TEE are processor extensions that can provide shielded code execution while guaranteeing the isolation, confidentiality and integrity of the computations.

Intel SGX is an instruction extension available on modern x86 CPUs manufactured by Intel. Similarly to ARM Trustzone [Azab et al., 2014] or Sanctum [Costan et al., 2016], 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 [Costan and Devadas, 2016]. 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. The Trusted Computing Base (TCB) of an SGX enclave is composed of the CPU itself, and the code running within. The TCB constitutes the surface of attack and is depicted in Figure 2.1. Differently, when not using Intel SGX, the system exposes a wider surface of attack that also includes the operating system, the bios or the hypervisor.

Attesting Intel SGX.

Intel provides a way for enclaves to attest each other. After the attestation process, enclaves will be sure that each other is running the code that they are meant to execute.

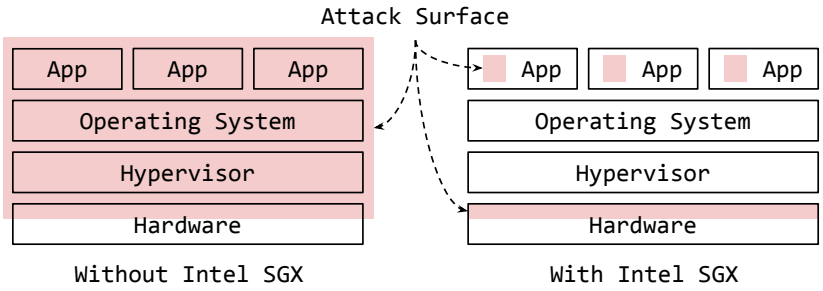


Figure 2.1 – System attack surface without (left) and with (right) Intel SGX.

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 [Costan and Devadas, 2016]. 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.

Shortcomings of Intel SGX.

Intel SGX comes with a run-time performance overhead, 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).

Moreover, one cannot rely on widespread adoption of such enabling technology. Instead, one needs to consider the end-users heterogeneity, including various microprocessor architectures, mobile users or even Internet of things (IoT) devices.

Finally, SGX enclaves were recently documented as being potentially subject to side channel [Lee et al., 2017] or speculative execution [Van Bulck et al., 2018] attacks. We see these attacks as out-of-scope within our work. We consider that they do not dismiss the concept of TEEs in general and will be addressed by evolution of the concept's implementation.

2.3 Group Sharing Model

Users and Administrators

We separate the end-users of our group-based data sharing system into *users* and *administrators*.

Users are humans or software agents and are uniquely identifiable. Users are organized into uniquely named *groups*. A user can be a member of multiple groups. Users that are part of a group are called *active* users, while the ones that are no longer part of a group are denoted as *revoked* users.

Administrators are privileged users managing group membership – they decide who joins and leaves the group. Users can discriminate the administrators among the fellow users. Administrators perform three operations : *create* a group, and *add* or *remove* a user from the group. To capture group membership, administrators define access control policies.

Access Control Policies

Access Control Policies specify the group members and their capabilities – *read* or *write*. Conceptually, these policies are very similar to Access Control Lists (ACL), a popular way of specifying access capabilities in file systems.

Some examples of access control policies for a group G are :

$$acl(G) = \{\text{ Alice, Bob, Carol }\} \tag{2.1}$$

$$acl(G) = \{\text{ Dave}:\texttt{read, write; Eve}:\texttt{read; Frank}:\texttt{read }\} \tag{2.2}$$

$$acl_{\texttt{read}}(G) = \{\text{ Dave, Eve, Frank }\} \tag{2.3}$$

$$acl_{\texttt{write}}(G) = \{\text{ Dave }\} \tag{2.4}$$

Example 2.1 shows a group in which all three users have identical capabilities. Example 2.2 illustrates a group in which only user Dave can create content while everybody else can only read. This example can be re-written by dividing the group per capability – Examples 2.3 and 2.4. Moreover, if Dave is the creator of the data, we call him sometimes the `owner` of the data.

Such a simple yet powerful access control model has found wide adoption with public cloud providers. For example Dropbox uses *editor*, *viewer*, and *owner* capabilities, while Amazon S3 specifies permissions as *read*, *write* and *full control*. Group based access control is also present in social networks (e.g., Facebook Groups) or instant communication (e.g., Whats-up).

Our group access control policy is a simplification of Role-Based Access Control (RBAC) [Sandhu, 1998]. RBAC is designed to capture capabilities that represent generic actions within an organization, and not necessarily data level access. Other access control models allow more complex policies, for example Attributed-Based Access Control (ABAC) [Hu et al., 2015] assigns attributes to users and data, and then grants access if attributes match. We argue our choice of policy representation due to its simplicity and popularity.

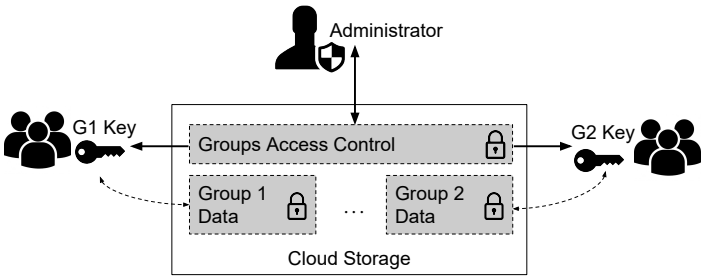


Figure 2.2 – Cryptographic Group Access Control.

The Group Key `gk`.

Within the *end-to-end security* paradigm the data is encrypted before being stored on the cloud. To make sure that different groups do not comprehend each others data, each group employs a different encryption key. We refer to this encryption key generically as the group key `gk`. Intuitively, at any given time only the *active* users should posses the knowledge of `gk` and not the *revoked* users, nor the cloud storage.

The group access control policies $acl(G)$ therefore specify who can access and with what capability the `gk`. Figure 2.2 illustrates two groups of users, each with a different group key `gk`, allowing them access to two distinct group data-sets.

Confidential Distribution of `gk`.

We require that `gk` is cryptographically known only to *active* users of the group and to nobody else. We denote by *key enveloping* the process of wrapping `gk` in an envelope with the property that only *active* members of the group can open the *envelope*. It should be cryptogaphically infeasible to the revoked users or the cloud storage to open this envelope. Building the envelope represents therefore a *key distribution* mechanism that offers *confidentiality* guarantees (detailed in Subchapter 2.4). Of course, such a mechanism needs to satisfy the notion of *semantic security* (IND-CPA).

Since administrators perform policy changes, they also construct or change the key envelopes. The lifetime of an envelope is linked to the lifetime of the underlying key `gk`. Therefore if the `gk` changes – like in the case of a revocation – the envelope will change as well.

Anonymous Distribution of `gk`.

In some cases we want to make sure that users in the group do not know each other – in other words a receiver of the envelope should not infer who else is a receiver of the same envelope. In such a case, we need an *anonymous key distribution* mechanism (detailed in Subchapter 2.5). Differently than for the *confidential* distribution scheme, *active* users can

mount attacks in order to discover peer users, choosing ciphertexts at will for which they know the encryption key `gk`. We therefore require the *anonymous* key distribution scheme to be secure under the tougher Chosen Ciphertext Attack (IND-CCA) model.

Revocation of `gk`.

Whenever a user is evicted from an access policy it is imperative that a new group key `gk` is created and broadcasted to all the remaining *active* users in the group. The operation of changing the `gk` is called *re-keying*. Even though our model links a re-key to a revocation event, the same procedure could be applied when users join the group in order to maintain the secrecy of past shared data.

Within the context of revocation one can wonder about what happens to the data during a re-key. Within the model of *lazy* revocation [Kallahalla et al., 2003], data created after revocation is encrypted using the new `gk`, but existing data remains encrypted by the old `gk`. The scheme sets the lower bound in terms of computational costs, requiring no re-encryption. However, we argue that such a model does not offer the best possible security guarantees. Instead we require our model to satisfy a tighter security model, namely *active* revocation model. *Active* revocation imposes that all prior data be re-encrypted with the new group key.

We consider that users who entirely downloaded, decrypted and *viewed* a resource, can cache the plaintext of the resource locally. After revocation, these users can continue accessing their local copy, even if the remotely stored resource is replaced with a re-encrypted one. As such, the security guarantees of *revocation* target resources not entirely downloaded by revoked users while being active.

Remote Storage Assumptions.

We assume that the remote storage supports a simplified interface for resource download and upload, denoted by $get(id) \rightarrow d$ and $put(id,\ d)$ where id is the identifier of the resource while d is the data content. Sometimes, the cloud storage can also be used as a broadcasting interface for group access control changes. In such a model, 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.

Availability of TEE Assumptions.

We assume a limited availability of TEE for our system actors. Among end-users, we assume that only *administrators* are equipped with TEE namely Intel SGX capabilities, and not the regular users. Such an assumption is reasonable considering the small number of administrators compared to users. For example, even in a very large size organization, only a handful of employees (e.g., IT desk) administrate access control.

We assume that the cloud possesses limited TEE capabilities. Clients can outsource to the cloud TEE *data-locality* enabled tasks, which do not interfere with the cloud data delivery interface. As such, we consider that users can not use the cloud TEE to download or upload data by proxy-ing the data through the TEE. The performance limitations of SGX can not enable such a scenario, notably when aiming for high servicing throughput.

Threats Overview

Untrusted Cloud Storage. We assume that the cloud storage satisfies liveliness assumption and does not deny service. However, the cloud storage employs an untrusted hardware and software environment that can be exposed to an attacker. Such an attacker can be interested in getting access to the group data – thus breaking the *confidentiality* – or differently learning the composition of the group – breaking the *anonymity* guarantee. The preferred threat model in the literature of untrusted cloud storage emerges as *honest-but-curious* [Garrison et al., 2016]. Such a model specifies that the cloud storage plays by the protocol but it's not trusted for knowing the data shared by the users. Moreover, similarly to related work [Bacis et al., 2016] the cloud storage is not trusted with performing the access control to the data.

Revoked Users. Users that have been revoked or are external to the system behave arbitrarily. They try to discover shared content or sometimes group members identities. To do so, they can intercept, decipher and alter exchanged messages (i.e., *Dolev-Yao* adversarial model [Dolev and Yao, 1983]). Also, as the cloud storage is not trusted for performing access control, revoked users can recuperate remotely stored content at will.

Active Users. We consider that group members are trusted for keeping the `gk` secret while they are *active*, and thus preserving *confidentiality*. Even though, intuitively, *active* group members are an entity of trust, it is not always the case. We consider two cases of *active* users manifesting an illicit behavior. First, in order to break the *anonymity* guarantee, active users might behave in an untrusted manner to find who else is a member of the group. Second, group members could try to *over-provision* as much sensitive cryptographic materials (e.g., keys) as possible while *active* with the intent of making use of them later once revoked.

Out of Scope. We consider out-of-scope hiding the size of groups, how often members communicate and the size of the content that they exchange. We also consider that the untrusted storage does not keep a history of versions of all the stored data, but only keeps the latest version – a reasonable assumption considering the very large volume of data and the storage cost that this would entail [Bacis et al., 2016].

2.4 Confidential Key Distribution

We describe next *state-of-the-art* enveloping methods for distributing a group key `gk` with *confidentiality* guarantees.

Hybrid Encryption (HE)

Suppose that we want to come up with a simple, yet secure, enveloping scheme for a group key `gk`. We can make use of an asymmetrical encryption primitive [Ferguson and Schneier, 2003], 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. The ciphertext resulted from this encryption constitutes the key envelope. Users of the group can then deduce `gk` by decrypting the fragment of the envelope that was encrypted with their public key by using their private key. In order to point the users to their ciphertext fragment in the envelope, one can include a mapping from the user identity to the ciphertext fragment. This construction is sometimes referred to as Hybrid Encryption (HE) [Garrison et al., 2016], or Trivial Broadcast Encryption Scheme [Stinson, 2005]. HE is utilized as an access control methodology for Windows Encrypted File System (WinEFS) or the Pretty Good Privacy (PGP) [Zimmermann, 1995] program, used for cryptographic protection of file or emails,.

However, HE comes with a number of weaknesses. First, the size of the key envelope grows linearly with the number of members in the group, making it impractical in the context of very large groups (as seen later in the final part of the sub-chapter). Second, during revocation, when a new key `gk` needs to be created; the entire envelope 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 key envelope on the cloud storage will also seriously expand.

Identity-Based Encryption (IBE).

Moreover, 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 Infrastructure [Ferguson and Schneier, 2003] can be used to solve this issue. Besides the trust risks that the PKI brings [Ellison and Schneier, 2000], one needs to account for the practical costs of setting up, running and accessing PKI. To mitigate these risks, one could choose to substitute public-key primitives with identity-based ones. Identity-Based Encryption (IBE) [Boneh and Franklin, 2001; Waters, 2005] 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. Obviously, both Hybrid Encryption with PKI (HE-PKI) and Hybrid Encryption with Identity-Based Encryption (HE-

IBE) have the same inner functioning, when making abstraction of the key methodology choice.

Broadcast Encryption (BE)

Broadcast Encryption [Fiat and Naor, 1993] is a public-key cryptosystem with a unique public key that envelopes the entire system, contrary to the Hybrid Encryption scheme where each user uses a different public key. However, each user in a Broadcast Encryption (BE) system has a unique private key generated by a trusted authority. To randomly generate a group key `gk` and the associated envelope (named *encrypt* operation within BE systems), one makes use of the system-wide public key. On the other side, when users want to reveal `gk` (*decrypt* in BE systems), they make use of their individual private key.

As BE schemes come with different contextual models, we impose a number of conditions. First, to maintain the same threat model as Hybrid Encryption (HE), we are only investigating the use of fully collusion-resistant BE schemes [Boneh et al., 2005b], 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 [Delerablée et al., 2007]. Third, as in the case of HE, we would prefer constructions that can accommodate the use of Identity Based Encryption (IBE).

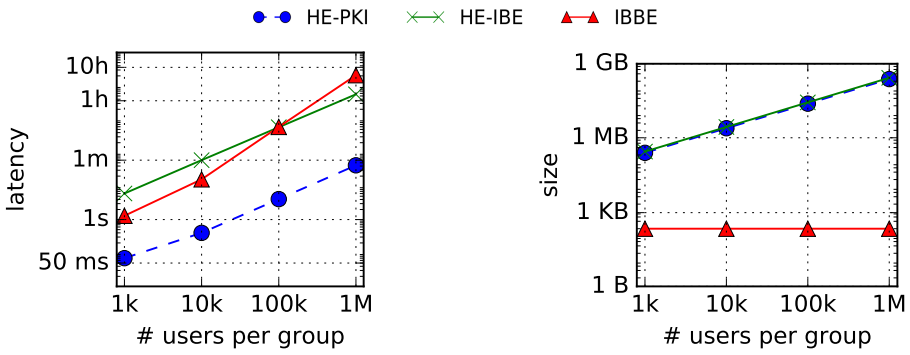
Identity-Based Broadcast Encryption (IBBE)

Piercing through the existing research literature, we identified an Identity-Based Broadcast Encryption (IBBE) scheme [Delerablée, 2007] that not only fulfills all the aforementioned requirements, but also operates with key envelopes 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 key envelope, the computational cost of IBBE might be excessive for practical use.

Inefficiency of Existing Key Management Schemes

Figure 2.3 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 key envelope. The optimality of IBBE regarding the size of the envelope is immediately obvious. It always produces 256 bytes, regardless of the number of users per group. That is preferable compared to HE-PKI and



(a) Latency for group creation.

(b) Group metadata expansion.

Figure 2.3 – Performance of HE-PKI, HE-IBE and IBBE.

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 in TEE is inadequate. In the course of a future chapter of this dissertation (Chapter 4), we describe two innovative propositions, one that changes the traditional assumption of the IBBE scheme, and a second that lowers the user decryption time.

2.5 Anonymous Key Distribution

We describe next *state-of-the-art* enveloping methods for distributing a group key `gk` not only considering *confidentiality* but also *anonymity* guarantee. In the presented form, none of the aforementioned schemes (HE-PKI, HE-IBBE, IBBE) guarantee the *anonymity* of participants in the scheme.

Preserving the anonymity of group members only against outside and not inside attackers has been proved easier to tackle [Fazio and Perera, 2012], however we aim for full anonymity against both peer members and outside attackers.

Anonymous Hybrid Encryption

In practical systems, PGP [Zimmermann, 1995] which relies on Hybrid Encryption (HE) for access control, addresses 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 (i.e., the *envelope*). 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).

Anonymous Broadcast Encryption (ANOBE)

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.* [Barth et al., 2006] (denoted hereafter BBW, per the authors' initials) achieves inner and outer anonymity. Their construction extends the public key enveloping model of PGP, by incorporating strongly unforgeable signatures [Boneh et al., 2006] such that an active attacker who is member of the group cannot reuse the envelope to broadcast arbitrary messages to the group, and thus achieving Indistinguishability under Chosen Ciphertext Attack (IND-CCA). 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 Diffie-Hellman (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.* [Libert et al., 2012] by suggesting the use of *tag-based encryption* [MacKenzie et al., 2004] to hint users to their ciphertext fragment. To the best of our knowledge, no practical system has integrated *tag-based encryption* in practice.

As both anonymous schemes discussed above make use of Hybrid Encryption (HE), their performance is lower bounded by the performance of HE. In the same time, the previous chapter (Figure 2.3) argued the inefficiency of HE which transitively applies to the anonymous schemes. We require therefore the exploration of alternative anonymous constructions that can scale to cloud specific access control workloads. Within a later section of this dissertation (Chapter 5) we present an innovative ANOBE design that can leverage TEE for only a subset of operations to achieve practical performance.

2.6 Data Revocation

If the two prior sub-chapters investigated the existing methods for enveloping `gk` for *active* users, the current sub-chapter focuses on what happens to the data during a revocation. Active revocation offers the best security guarantees by performing a full re-encryption of all the data resources upon a revocation. However, active revocation

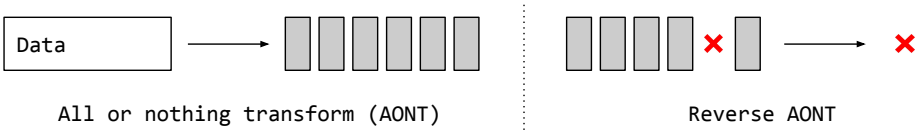


Figure 2.4 – AONT Scheme.

AONT is reversible if and only if all transformation output is known.

schemes are highly impractical due to their high processing and data access costs. We focus next on existing techniques for achieving practical active revocation.

Super Blocks.

An alternative approach is to implement active revocation with the re-encryption of only a small subset of the data. The resource (e.g., data file) is split in fixed-size blocks. All these blocks are necessary in order to reconstruct the file at the client side. Only a few of the blocks however, are re-encrypted using the new `gk` upon a revocation. These blocks are called *super-blocks*. Other blocks remain encrypted with the original key. In order to prevent *pre-provisioning* attacks – in which active users pre-provision sensitive material to make use of after revocation – it is necessary to enforce that all blocks be necessary to reconstruct the file but also to possibly determine which are the super blocks. To address these inconveniences one can make use of All or Nothing Transform detailed in the following.

All or Nothing Transform (AONT).

The AONT [Rivest, 1997] transformation is reversible only if *all* of its resulting output is known (Fig. 2.4). In addition, any incomplete sub-set of this output reveals nothing about the input data. Rivest introduced AONT for hardening against key determination brute force attacks for block ciphers, by having them performed on the whole ciphertext rather than on a single block [Rivest, 1997]. The transformation employs a symmetric encryption coupled with the hashing of each encrypted block. All encrypted block hashes are *xor*-ed with the symmetric key, resulting in a small packet that we commonly refer to as *tail* and that is appended to the output of the transformation. AONT is also used for defining Optimal Asymmetric Encryption Padding (OAEP) [Boyko, 1999] and is standardized as a padding technique for RSA [Kaliski and Staddon, 1998].

Intuitively, the use of AONT can significantly lower the processing cost of an active revocation scheme by requiring to re-encrypt only the tail and not an entire file. It is however sensitive to *pre-provisioning* attacks, which can considerably augment the power of the attacker under our threat model. A malicious user can indeed selectively pre-provision the sensitive information that are the *tail* blocks. In a second phase, once revoked, this user

could download the remaining blocks and decrypt them using the previously downloaded tail. Our goal is instead to build a re-encryption scheme that is robust to a curious user arbitrarily provisioning sensitive information.

Data Locality and TEE.

While security is the most important aspect for active revocation, the data locality of the re-encryption operation has a strong impact on performance, which may in turn increase the duration of the revocation operation and increase the power of a malicious revoked user. A naive solution would be to download all data blocks subject to re-encryption at the data owner side, re-encrypt them, and upload them again to the cloud. This approach satisfies confidentiality requirements, but it is impractical for large data sets.

Instead, under the assumption that the cloud storage is equipped with TEE we can require that re-encryption is performed on-site, as close as possible to the data. TEEs are becoming commonplace in public cloud IaaS offerings [Russinovich, 2017]. However, the sole use of a TEE for active revocation is not sufficient to reach efficiency. Re-encrypting the entirety of the data within one or more SGX enclaves is as impractical as rekeying at the data vendor. Within Chapter 6 we present an innovative construct that leverages the guarantees offered by TEE together with an AONT, to securely identify and re-encrypt only the super blocks without leaking these to the otherwise untrusted cloud storage or to malicious clients.

2.7 Open Challenges Summary

The conclusive driver of our work is the construction of efficient *end-to-end* remote storage systems, incorporating three cryptographically enabled pillars : *confidential* key distribution, *anonymous* key distribution and *practical* data revocation.

1. Confidential key distribution. State-of-the-art schemes HE-PKI, HE-IBE, IBBE are impractical for highly dynamic and large workloads (Figure 2.3). The schemes are either computational efficient but not storage footprint optimal, or the other way round. Our first open challenge is to find such a scheme that is both computational and storage efficient. To do so, we can use our instrument of efficiency : TEE availability for group *administrators*. A functional end-to-end system is required to prove the practicality of the approach.

2. Anonymous key distribution. State-of-the-art schemes guaranteeing anonymity (ANOBE) are as inefficient as their non-anonymous counterparts. As such, the open challenge is to find an efficient underlying anonymous scheme that can leverage TEE within a limited deployment and prove its efficiency within a realistic end-to-end system design.

3. Practical data revocation. Entirely re-encrypting data locally and then re-uploading it to the cloud is practically prohibitive. Efficient approaches leverage All-Or-Nothing transform (AONT), in order to re-encrypt just a part of the resource. However, they do not lever-

age data-locality. The open challenge is therefore to find an efficient end-to-end system design incorporating TEE at the provider side and AONT to re-encrypt only a part of the data, while preventing malicious users to over-provision these data parts.

We address these three challenges in subsequent chapters. But first, next chapter surveys related work.

CHAPTER

3 Related Work

This chapter surveys related work. We give examples of applied cryptographic constructs and secure end-to-end systems that solve problems related to our context. If the previous Background Chapter 2 presented the building blocks of our contributions, this chapter solely exemplifies similar systems.

We present related work from three perspectives. First, we look into related cryptogrpahic schemes that are closely related to access control. Second, we survey a number of end-to-end systems that make use of Trusted Execution Environments (TEE) and notably Intel SGX. Finally, we survey end-to-end systems over untrusted storage that relate to our problem definition.

3.1 Applied Cryptography for Access Control

Besides the building blocks described by Chapter 2 (HE, IBBE, and ANOBE), related work contains a number of other applied cryptography constructs used in the context of access control.

Attribute Based Encryption (ABE) 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 [Goyal et al., 2006] and ciphertext-policy ABE [Bethencourt et al., 2007]. However, when employed for simple access control policies, such as our group sharing context, ABE has substantially greater costs than identity-based encryption [Garrison et al., 2016].

Hierarchical Identity Based Encryption (HIBE) [Boneh et al., 2005a] and **Functional Encryption** (FE) [Boneh et al., 2011] 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 unsuitable for practical use when relying on pairings [Fisch et al., 2017].

Proxy re-encryption [Ateniese et al., 2006] is a cryptographic system in which the owners of encrypted data can delegate the re-encryption of the data to a proxy, with the intent of sharing it with other users. 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 [Green and Ateniese, 2007], or with attribute-based encryption [Green et al., 2011; Sahai et al., 2012].

Secure Multicast. The related research area of multicast communication security [Stinson, 2005; Canetti et al., 1999] defines efficient schemes focusing exclusively on revocation aspects. Logical Key Hierarchy [Wallner et al., 1999] is a re-keying scheme in which communications for revocation operations are minimized to logarithmic sizes. Other schemes [Fiat and Naor, 1993; Naor and Pinkas, 2000] exploit a secret sharing mechanism, considering that no coalition of revoked users larger than a threshold number is trying to decrypt the transmission.

Diffie-Hellman Group Key Agreement. Communication secrecy can also be achieved by enabling a group of participants to interactively derive a group key. Such techniques are popular among communication systems (e.g., WhatsApp, Threema) by using Diffie-Hellman (DH) group key agreement and derivation [Rösler et al., 2018]. Such protocols require all active participants to contribute to the creation of the group key and achieve *confidentiality*. Pung [Angel and Setty, 2016] uses private information retrieval (PIR) in conjunction to a group DH key derivation to achieve *anonymity*. One should note that such a mechanism is different from our target model, in which users do not need to actively participate in the creation of the group key, no matter the number of groups they belong to.

Cryptographic Secure Deletion relates to access control as it is equivalent to revoking access to the last member of a group. A straw-man approach is to encrypt the data and then "*forget*" the encryption key [Boneh and Lipton, 1996]. For example, secure deletion by randomized keys [Peterson et al., 2005] – similarly to Hybrid Encryption (Section 2.4) – encrypts every file with a random symmetric key, and then over-encrypts the key with a second key. When a deletion happens, only the second key changes but not the inner key. This approach allows for fast deletions but does not satisfy our requirements. Malicious users can indeed provision all the file keys and then, once deleted, retrieve the encrypted content without requiring the outer key.

3.2 Systems using Intel SGX as TEE

Intel SGX has been extensively used in shielding applications and infrastructure platform services that handle sensitive data. We detail in the following a number of such systems.

VC3 [Schuster et al., 2015] relies on SGX as TEE to guarantee the *confidentiality* and *correctness* of MapReduce computations. If the confidentiality is tackled by requiring each *mapper* and *reducer* node to run computations in SGX, the correctness is achieved by aggregating small profs validated by an administrative *verifier* entity. Besides TEE, the approach relies solely on standard cryptographic constructs (Section 2.1). Moreover, as the system is open to external users to load their code in enclaves, run-time errors such as *unsafe memory access* can increase the attack surface. VC3 solves the issue by memory *read* and *write* invariants that keep access within the enclave's memory. Differently our model (Section 2) considers that the code running in TEE is entirely trusted once attested.

Iron [Fisch et al., 2017] is close to the thesis scope in the sense that it takes advantage of SGX to build a practical encryption scheme for an unpractical strategy thus far. 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.

SCBR [Pires et al., 2016]. At the level of infrastructure services, SCBR 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.

Hybster [Behl et al., 2017] relates to our goal with regards to the reduction of overhead for an otherwise costlier design. Hybster proposes a hybrid state-machine replication protocol. Hybster 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.

SecureKeeper [Brenner et al., 2016] addresses the confidentiality of ZooKeeper distributed coordination by also employing SGX. SecureKeeper targets a rich adversarial model, by guaranteeing confidentiality of state keys and values stored in ZooKeeper's tree like database. The system leverages an *entry enclave* communicating with end clients, and sharing a long term secret with the *worker* enclaves. A second enclave type is used to han-

dle specific ZooKeeper sequential naming per each leader replica. Differently than SecureKeeper, our model assumes that the storage provider is *honest-but-curious* and therefore respects the serviced protocols.

DelegateTEE [Matetic et al., 2018] proposes a mechanism to delegate access control for online services. Account *owners* can specify a group of additional users – *delegatees* – able to access the services on behalf of the *owners*, while not disclosing the secret credentials of the *owners*. The approach relies on TEE and notably Intel SGX encalves to proxy the requests *to* and *from* the target service. Two architectural choices are presented, a peer-to-peer one requiring TEE enabled delegatees, and a second centrally brokered design. Similarly to our goals, DelegateTEE considers two use cases : *confidential* and *anonymous* delegation guarantees.

ShieldStore [Kim et al., 2019] proposes a TEE enabled design guaranteeing the *confidentiality* of in-memory key-value stores. The current version of Intel SGX limits the memory size of the enclave page cache (EPC) to 128 mega-bytes, making it unsuitable for in-memory key-value stores that surpass this threshold. The approach proposes storing the underlying key-value data structure (i.e., a hash table with chaining lists) encrypted within the untrusted zone. During *get* operations the values are decrypted and validated for integrity within the enclave. Moreover the approach replaces the paging mechanism of Intel SGX with a finer grained custom implementation at the level of keys.

Libseal [Aublin et al., 2018] proposes an out-of-the-box replacement for the transport layer security (TLS) library that besides the functionalities of the latter produces service audit logs, capable of proving client or remote service violations of service level agreements (SLA). To do so, Libseal requires that the TLS connections are terminated in Intel SGX enclaves at the service provider side. To speed-up the TLS termination implementation, the authors propose a provisioning of batch memory allocations inside the enclave and storing the data sent through TLS outside the enclave. Differently than our model, Libseal targets only *integrity*, and not *confidentiality* nor *anonymity*.

X-Search [Mokhtar et al., 2017] uses TEE to enable private web search queries. Owning to Intel SGX, X-Search performs orders of magnitude better than state-of-the-art while offering stronger security guarantees. The system uses an Intel SGX enabled proxy. The users TLS their query to the proxy's enclave, who in turn adds obfuscation keywords to the query, while retaining the user query for future obfuscation use. Upon receiving the obfuscated query result from the web provider, the proxy filters and forwards the actual result to the user. The *unlinkability* property targeted by X-Search is identical to the sender and recipient *anonymity* of our model. However, X-Search's *indistinguishability* property is relevant within the web search scenario and does not apply in our cloud storage use case. If in the web search scenario the provider *sees* the user plaintext but can not distinguish it as genuine, our model requires that the provider never *sees* the data (i.e., *confidentiality*).

3.3 Remote Untrusted Storage Systems

We review a number of systems that similarly to our model consider the storage provider *untrusted*. Even tough our model (Section 2) targets the properties of *confidentiality*, *anonimity*, *cryptographic access control*, and *revocation* we extend our survey of related work to systems that include properties such as : *consistency*, *freshness*, *de-duplication*, and *dispersal* methods. *Consistency* states whether the latest write to the data is immediately (i.e., *strong* consistency) or *eventually* recorded. *Freshness* on the other side requires *read* operation to receive the latest recorded value protecting from an accidental or adversarial roll-back. *De-duplication* consists of storing only once two or more identical logical data blocks. Data *dispersal* consists of transforming the data in many shares such that a subset of shares can reconstruct the data.

SCFS [Bessani et al., 2014] is a *"shared cloud-backed file system"* interfacing user end-points with multiple cloud storage, optionally providing *confidentiality* guarantees for user data. The clients have *strong consistency* and POSIX semantics (i.e., file system specific) while being backed by *eventually consistent* remote storage. The system makes use of a trusted coordination layer that stores file metadata including access control and a locking service for concurrent writes. To optimize operations SCFS uses a caching component on client side and cleans old versions of data based on administrative policies. For *confidentiality*, files are encrypted and the key is dispersed on multiple non-colluding servers [Shamir, 1979]. SCFS trusts the infrastructure hosting the access control monitor and a sub-set of the remote storage containing the encryption keys. Differently, our goal is to develop access control mechanisms *cryptographically*, undecipherable to the possibly adversarial service provider.

CloudProof [Popa et al., 2011a] similarly considers the cloud storage untrusted and provides *confidentiality*, *integrity*, *consistency* (i.e., a total order on write operations), and *freshness* (i.e., reads service latest write) guarantees. The access control is implemented cryptographically by enveloping a symmetric key and a signing key for read and write capabilities. The enveloping is done using Broadcast Encryption (BE) [Boneh et al., 2005b]. Changes in groups membership trigger a *lazy revocation* method – data is re-encrypted with the new group key during the first read. In order to guarantee *consistency* and *freshness*, the system makes use of attestations issued by both client and cloud operations. Attestations are chained by a hash-and-sign mechanism, making it possible to iterate the chain backwards to observe correctness. Differently than our threat model which considers the cloud *honest-but-curious*, owing to the attestation mechanism CloudProof targets an enriched *entirely untrusted* cloud provider. However, BE and notably it's identity version IBBE do not offer satisfactory performance for large workloads (Figure 2.4).

Tahoe [Wilcox-O'Hearn and Warner, 2008] is a cloud backed file-system running on untrusted clouds. Tahoe error encodes the encrypted data and disperses the shares over multiple storage. Tahoe extends *read* and *write* data access capabilities by a *verification* capa-

bility, which validates that data is un-modified. If it's the case, verification can efficiently point the compromised sections due to a Merkle Tree built over the ensemble of shares. Write capabilities are introduced through asymmetric key pairs. Freshness is achieved by validating the latest version by a quorum. Tahoe, however, does not explain how these capability keys – which are used for protecting the resources similarly to our `gk` (Section 2.3) – are distributed among users. As such, there are no assumptions of a cryptographic key distribution mechanism.

Sieve [Wang et al., 2016] platform allows users to store data encrypted in the cloud and discretionary delegate access to the data to third party web services. Sieve makes use of attribute based encryption (ABE) for access control policies. During revocation a key homomorphism [Boneh et al., 2013] is used for re-encrypting the whole data at the provider side. It should be noted that ABE is slower than the public key operations of HE (our baseline – Figure 2.3) while the employed additive key homomorphism is orders of magnitude slower than symmetric encryption. Utilizing the two concepts for cryptographic access control does not fit with our scalability goal of supporting very large user bases and data volumetry.

Reed [Li et al., 2016a] reconciles the procedure of *re-keying* (i.e., changing the key as during a revocation) and out-sourcing to remote storage encrypted de-duplicated data. Similar to our target context, Reed regulates access control in a cryptographic way. They chose Attribute Based Encryption (ABE) [Bethencourt et al., 2007] – which, as illustrated by their evaluation, is suitable only for hundreds of users (see Figure 8.a. of [Li et al., 2016a]) – and therefore not fit for very large user volumetry. Reed enables de-duplication and re-key by using *convergent encryption* (i.e., using the hash of the data as the encryption key) and an all-or-nothing transform (Section 2.6). During re-keys only a small (64 bytes) package is re-encrypted. However, under our model (Section 2.6), provisioning this package is as harmful as provisioning the encryption key.

Oceanstore [Kubiatowicz et al., 2000] proposes a *global-scale* shared storage system over untrusted infrastructure. The system is enabled with confidentiality by symmetric encryption. For *read* access a user needs to be in possesion of the encryption key while for *writes* the server checks an attached signature. Note that there is no mention of how these keys are distributed to users – therefore not considering *access control*. Ocenastore further proposes a distributed routing solution for locating resources, leveraging Bloom Filters for efficiency. The system provides a *deep* archival functioning achieved by erasure codes. Explicitly stated as "*under development*", it is unclear if such a system can be practically efficient to meet its ambitious goals. The project lacks a benchmarking phase, while no further update report was issued by the authors.

CDStore [Li et al., 2016b] proposes the design of a distributed storage solution that targets both reliability and security by employing an all or nothing transform (AONT) (Section 2.6) and then error encoding the data over multiple providers. As such, the providers

can not recover the original data unless they collude. The solution was designed to target backup scenarios – thus write dominated workloads – where the concept of de-duplication can bring considerable savings. To re-conciliate data disperssion and de-duplication, the authors propose the use of *convergent encryption* where the key is the actual hash of the data. As convergent encryption is prone to *side channel attacks* (i.e., guessing the plaintext based on an identical hash), client application only de-duplicates the user data, and not the one of other users. Further, at server side, inter-user de-duplication is enabled, assuming that cloud does not collude with compromised users. Differently than CDStore, in our model we do not target *confidentiality* guarantees by non-colluding assumption of multiple cloud storage. Instead we require it to be enabled cryptographycally by a key management scheme.

UniDrive [Tang et al., 2015] similarly relies on multiple clouds to achieve reliability and security, with a focus on client observed performance. The proposed solution is solely client side, without a proxy-like coordination layer or an agent running at cloud storage. Files metadata are symmetrically encrypted while data is error encoded with a *non-systematic* code over multiple clouds. One should note that if symmetric encryption provides *semantic security*, using a non-systematic code does not – the generative matrix used by the latter can be easily obtained when both plaintext and ciphertext are known. The system provides de-duplication and improves performance by over-provisioning parity blocks on clouds with faster network links. The system makes no assumption of handling the encryption keys. UniDrive does not consider sharing operations among users – as such no access control mechanism is discussed.

Cyrus [Chung et al., 2015] proposes a client-defined architecture targeting security and reliability of remotely stored data, with a focus on the heterogeneity of cloud providers. Rather than using a centralized metadata layer Cyrus disperses the metadata – similarly to the data – on many clouds. Cyrus considers identical providers that use the same hosting infrastructure (e.g. Dropbox similarly to S3 relies on Amazon Datacenters) a colluding group. Therefore, it infers this relationship among providers and uses it when dispersing the data. Similarly to UniDrive, Cyrus uses a non-systematic erasure code to generate the data shares, however it seeds the generative matrix by the user secret key to obtain user independent transformations. The system keeps track of cloud storage selection to learn and optimize future availability times. The APIs of Cyrus do not incorporate access control.

Dep-Sky [Bessani et al., 2013] proposes leveraging multiple cloud storage for reliably and securely storing user data. The work particularly focuses on addressing client *read* and *write* operations by a *quorum* mechanism within the byzantine setting – meaning that if f clouds are malicious then $3f+1$ clouds are necessary. The system proposes a *locking* mechanism for concurrent writes. To enable confidentiality the data is encrypted while the keys are dispersed by a secret sharing scheme [Shamir, 1979] over many cloud storage. Even

though this latter approach mitigates key management, it is unsure how access control could be adopted by the system without an additional trusted component.

Iris [Stefanov et al., 2012] address the outsourcing of enterprise class file systems to the cloud, guaranteeing data *integrity* and *freshness*. Clients communicate with a middle layer, proxy-ing requests to the cloud storage. The proxy sits in a trusted environment while the clouds have a byzantine behavior. The proxy ensures integrity by constructing Merkle Trees and periodically issuing *proofs of retrievability* (POR) against the cloud storage to ensure data freshness. The proxy is reliably distributed over multiple nodes by using `memcached`. Compared to our requirements, Iris does not guarantee *confidentiality* and *anonymity*.

3.4 Closing Remarks

Even though much related work focused on untrusted storage systems, no conclusive work addressed cryptographic access control of end-to-end secure systems for *highly large and dynamical workloads*. Designing such a system would need first re-visiting the underlying cryptographic access control constructs.

We reiterate that the previous Backround Chapter 2 listed three open challenges (Section 2.7) in the context of the thesis : creating efficient (1) *confidential* and (2) *anonymous* access control extensions, and (3) an efficient *revocation* capability. The following three chapters will each address one of these problems. They will introduce novel cryptographic schemes and incorporate them in end-to-end systems – much like the ones surveyed within this chapter – to prove their efficiency.

CHAPTER

4

CON-SKY: Confidential Key Distribution with TEE

State-of-the-art schemes for confidential key management are impractical for highly dynamic and large workloads. This chapter presents a novel key management technique that leverages trusted execution environments (TEE) for the minimal set of *administrative* operations. Such change of assumptions permits deriving a computational complexity cut for Identity-Based Broadcast Encryption (IBBE) from quadratic to linear for the key enveloping operation. To speed up the user de-enveloping operation we propose a partitioning mechanism, splitting the group into sub-groups and thus bounding the user computational effort to the partition size. Finally, we detail an end-to-end system for confidential key distribution, namely CON-SKY.

The innovative concepts presented within this chapter have been published in the *International Conference on Dependable Systems and Networks* (DSN) [Contiu et al., 2018a].

Within the *end-to-end* security paradigm a confidential data sharing scheme consists of making available a group key `gk` exclusively to the *active* members of the group. To do so, the key is *enveloped* within a ciphertext that can be publicly transmitted across the untrusted medium, the cloud in our case. In the earlier Chapter 2 we observed that existing methods for enveloping `gk` are inefficient for highly dynamic and large group memberships. For example, Hybrid Encryption with Public Key Infrastructure (HE-PKI) or Identity-Based Encryption (HE-IBE) are inefficient mostly due to the linear growth of the envelope size (Figure 2.3). Differently Identity-Based Broadcast Encryption (IBBE) produces a small constant size envelope but has an impractical computational cost.

In order to achieve a practical confidential sharing scheme, we retain the small storage benefit of IBBE and derive a complexity cut for its computational time. We leverage an instrument of efficiency : we consider that *administrators* are equipped with SGX capabilities. As such, *administrators* besides performing group membership changes, hold a

secured SGX enclave that acts as a Trusted Authority (TA). Adapting IBBE scheme to such a change of assumption gives birth to a new scheme that we name IBBE-SGX.

We build an end-to-end confidential sharing system CON-SKY that makes use of IBBE-SGX. To further optimize the end-users computational time, CON-SKY makes use of a partitioning mechanism. Finally we deploy CON-SKY over an untrusted storage and prove its practicality through micro- and macro- benchmarks (Section 7.1.1 and 7.2.1).

4.1 Key Management in CON-SKY

We will describe key management methodology of CON-SKY within three steps: (i) trust establishment and private key provisioning (Section 4.1.1); (ii) group key provisioning (Sections 4.1.2 and 4.1.3); and (iii) membership changes and key updates (Section 4.1.4).

4.1.1 Trust Establishment

CON-SKY makes use of Identity-Based Broadcast Encryption (IBBE). 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 CON-SKY 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 2.2).

Figure 4.1 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 mea-

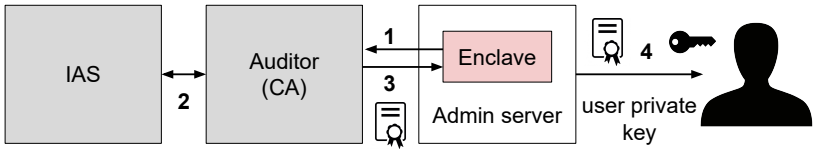


Figure 4.1 – CON-SKY initial setup.

surement 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).

4.1.2 Group Key Provisioning

Traditionally, the IBBE scheme [Boneh et al., 2005b; Delerablée et al., 2007] 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 Key.* The TA then uses the Master Secret Key M_{SK} to extract the secret key U_{SK} for each user U.
3. *Envelope Group Key.* The broadcaster generates a randomized group key `gk` for a given set of receivers S (i.e., the *acl* of the group), by making use of PK. Together with `gk`, the operation outputs the *envelope* as a public broadcast ciphertext c. The broadcast ciphertext can be publicly sent to members of S so they can derive `gk`.
4. *Unveil (or De-envelope) Group Key.* Any member of S can discover `gk` by performing the de-envelope operation given the secret key U_{SK} and (S, c).

Traditional IBBE requires a Trusted Authority (TA) to perform the *System Setup* and *Extract User Secret Key* operations. We rely on SGX enclaves as a TA design choice. Therefore, the master secret key *MSK* used by the two aforementioned operations is made available in plaintext exclusively inside the enclave, and securely sealed if stored outside of the enclave for persistence reasons.

Change of Assumption.

Within traditional IBBE the *Envelope Group Key* and *Unveil Group Key* operations rely on the system public key PK, and are thus usable by any user of the system. But differently than traditional IBBE usage scenario, our model requires that all group membership changes - generating the group key and envelope — are performed by an administrator. Therefore similarly to setting up the system and extracting user keys, administrators can make use of the MSK for the *Envelope Group Key* operation. The de-enveloping operation, however, remains identical to the traditional IBBE approach, being executed by any arbitrary user. We call an IBBE scheme adopting this change of assumption as IBBE-SGX.

IBBE-SGX vs. IBBE

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 enveloping operation drops from $O(|S|^2)$ for IBBE to $O(|S|)$ for IBBE-SGX, where $|S|$ is the number of users in the group. The reason behind the complexity drop is bypassing a polynomial expansion of quadratic cost, necessary in the traditional IBBE assumptions. We argue that this complexity cut is sufficient to tackle the impracticality of the IBBE scheme emphasized earlier in our preliminary benchmarks (see Figure 2.3). Second, by relying on MSK, one can build efficient access control specific operations, such as adding or removing a user from a group. IBBE-SGX can accommodate $O(1)$ complexities for both operations.

Unfortunately, IBBE-SGX maintains an $O(|S|^2)$ complexity for the user de-enveloping operation, during which, similarly to IBBE enveloping, the algorithm performs a polynomial expansion of quadratic cost. We address this drawback by introducing a partitioning mechanism as described later in Section 4.1.4.

Finally, we consider a re-keying operation, for optimally generating a new `gk` and envelope when the identities of users in the group S do not change. The operation can be performed in $O(1)$ complexity for both IBBE and IBBE-SGX.

4.1.3 Formal Specification of IBBE-SGX

Similarly to IBBE [Delerablée, 2007], IBBE-SGX 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.* provide a thorough overview of bilinear maps usage within the cryptographic setting [El Mrabet and Joye, 2017]. Moreover, the IBBE scheme implies the public knowledge of a cryptographic hash function $\mathcal{H}$, that differently than the hash methods described in Subsection 2.1 maps user identity strings to values in $\mathbb{Z}_p^*$.

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 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.

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 :

$$U_{SK} = g^{(\gamma+\mathcal{H}(u))^{-1}} \tag{4.1}$$

Envelope Group Key

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

$$\mathtt{gk} = v^k \tag{4.2}$$

The envelope, i.e., the group ciphertext (C_1, C_2) is then constructed by:

$$C_1 = w^{-k} \tag{4.3}$$

$$C_2 = h^{k \cdot \prod_{u \in S}(\gamma+\mathcal{H}(u))} \tag{4.4}$$

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:

$$C_2 = \left(\left(h^{\gamma^n}\right) \cdot \left(h^{\gamma^{n-1}}\right)^{\mathcal{E}_1} \cdot \left(h^{\gamma^{n-2}}\right)^{\mathcal{E}_2} \cdot \ldots \cdot \left(h^{\gamma}\right)^{\mathcal{E}_{n-1}}\right)^k \tag{4.5}$$

where :

$$\begin{aligned}
\mathcal{E}_1 &= \textstyle\sum_{u \in S} \mathcal{H}(u) \\
\mathcal{E}_2 &= \textstyle\sum_{u_1, u_2 \in S, u1 \neq u2} \mathcal{H}(u_1) \cdot \mathcal{H}(u_2) \\
\mathcal{E}_3 &= \textstyle\sum_{u_1, u_2, u_3 \in S, u1 \neq u2 \neq u3} \mathcal{H}(u_1) \cdot \mathcal{H}(u_2) \cdot \mathcal{H}(u_3) \\
&\ldots \\
\mathcal{E}_{n-1} &= \textstyle\prod_{u \in S} \mathcal{H}(u)
\end{aligned}$$

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 4.4. It thus requires a linear number of operations.

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

$$C_3 = h^{\prod_{u \in S}(\gamma+\mathcal{H}(u))} \tag{4.6}$$

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

Unveil (or De-envelope) Group Key

The de-envelope operation is executed identically for IBBE and IBBE-SGX, and relies on PK. A user can make use of the secret key u_{SK} to compute $\mathtt{gk}$, given $(\mathcal{S}, \mathcal{C})$

$$\mathtt{gk} = \Big(e(C_1, h^{\mathcal{P}}) \cdot e(U_{sk_r}, C_2)\Big)^{\left(\prod_{u \in S \setminus \{r\}} \mathcal{H}(u)\right)^{-1}} \tag{4.7}$$

$$\mathcal{P} = \frac{1}{\gamma} \cdot \Big(\prod_{u \in S \setminus \{r\}} \mathcal{H}(\gamma + u) - \prod_{u \in S \setminus \{r\}} \mathcal{H}(u) \Big)$$

Similarly to enveloping a group key for IBBE, the decryption algorithm requires a polynomial expansions to compute the exponent $\mathcal{P}$, thus bounding the complexity to a quadratic number of operations in $O(|S|^2)$.

Add User to Group 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 $\mathtt{gk}$. 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:

$$C_2 \leftarrow (C_2)^{\gamma + \mathcal{H}(u_{add})} \tag{4.8}$$

Remove User form Group Key

Whenever removing a user u_{rem}, all group elements $\mathtt{gk}, \mathcal{S}$ and $\mathcal{C}$ need to change. $\mathtt{gk}$ and C_1 can be computed by Formulas 4.2 and 4.3, once a new random value for $k \in \mathbb{Z}_p^*$ is picked.

Within the traditional assumption, C_2 is computed similarly to enveloping 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:

$$C_3 \leftarrow (C_3)^{(\gamma + \mathcal{H}(u_{rem}))^{-1}} \tag{4.9}$$

$$C_2 \leftarrow (C_3)^k \tag{4.10}$$

Re-key Group Key

Sometimes, it is necessary to change the value of $\mathtt{gk}$ 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.

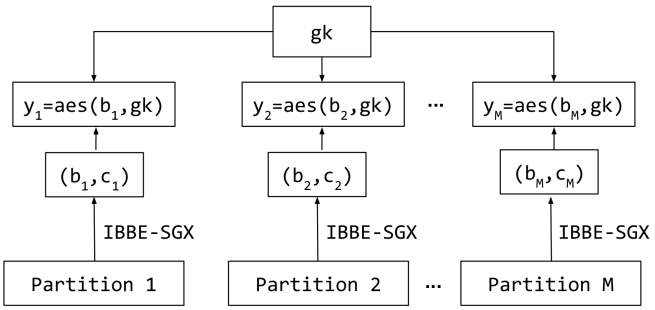


Figure 4.2 – Partitioning mechanism using IBBE-SGX and AES to protect the group key gk.

First, a new random $k \in \mathbb{Z}_p^*$ is generated and the new key computed by Formula 4.2. C_1 can be computed by Formula 4.3, while C_2 is computed from C_3 by Formula 4.10.

4.1.4 Partitioning Mechanism

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

As the de-enveloping is bound to the number of users in the receiving set, we split the group into partitions (sub-groups) and therefore limit the user de-enveloping time to the number of members in a single partition. Moreover, each partition group key wraps the prime group key gk, so that members of different partitions can communicate by making use of gk. The partition mechanism is depicted in Figure 4.2. The first step consists in splitting the group of users in fixed-size partitions. The *administrator* can then use the envelope functionality of IBBE-SGX to generate a sub-group (broadcast) key b_k and its envelope 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 key b_k as the symmetric encryption key. The entire group envelope of IBBE-SGX is therefore represented by the set of all pairs composed of the partition envelope and the encrypted group key (i.e., (c_i, y_i) in Figure 4.2). The untrusted cloud storage can then publicly receive and store this entire group envelope. Whenever a membership change happens, the *administrator* will update the list of group members and send the affected partition envelope to the cloud. The clients, in turn, can detect a change in their group by listening to updates in their envelope.

Partitioning Implications.

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 [Delerablée, 2007], results that the

Table 4.1 – 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 [Delerablée, 2007]
System Setup	$O(\|p\|)$	$O(\|S\|)$
Extract User Key	$O(1)$	$O(1)$
Envelope Group Key	$\|P\| \times O(\|p\|)$	$O(\|S\|^2)$
Add User to Group	$O(1)$	
Remove User from Group	$\|P\| \times O(1)$	
Unveil (or De-envelope) Group Key	$O(\|p\|^2)$	$O(\|S\|^2)$

public key for the IBBE-SGX scheme is linear in the maximal number of users in a partition (denoted 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 4.1. Enveloping a group key becomes the cost of enveloping 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 de-enveloping 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. Compared to IBBE when considering a single partition, the footprint is augmented by the symmetrically encrypted partition key (i.e., y_i) and the *nonce* required for this symmetric encryption. When considering an entire group, the cost of storing the group envelope 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 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 de-enveloping the group key. A small partition size reduces the de-enveloping

time while a larger one reduces the number of operations performed by the administrator to run IBBE-SGX and to maintain the metadata.

4.2 Operations Design

The operation for creating a group key is described in Algorithm 1. Once the fixed-size partitions are determined (line 1), the execution enters the SGX enclave (lines 2 to 7) 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 8).

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 group 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 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.

Removing a user from a group (Algorithm 3) proceeds by removing the user from the partition (lines 1 and 2). Next, a new group key is randomly generated (line 3). The former user hosting partition key and envelope 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 key and ciphertext that envelopes the new group key (lines 6 to 9). After sealing the new group key (line 10), the changes of

Alg. 1 CON-SKY Create Group

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

1 $\mathcal{P} \leftarrow \{\{u_1, ..., u_m\}, \{u_{m+1}, ..., u_{2m}\}, ...\}$
`begin ecall`
2 `gk` ←RandomKey()
3 **for** all $p \in \mathcal{P}$ **do**
4 $(b_p, c_p) \leftarrow sgx\_ibbe\_create\_partition(\texttt{MSK}, p)$
5 $y_p \leftarrow E_{h(b_p)}(\texttt{gk})$
6 $sealed\_gk \leftarrow sgx\_seal(\texttt{gk})$
7 **endfor**
`end ecall`
8 Store: (1) $sealed\_gk$; (2) $\forall p \in P : \langle \forall u \in p, y_p, c_p \rangle$

Alg. 2 CON-SKY 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 $p_{add} \leftarrow \{u_{add}\}$
`begin ecall`
$(b_{add}, c_{add}) \leftarrow sgx\_ibbe\_create\_partition(\texttt{MSK}, p_{add})$
4 $\texttt{gk} \leftarrow sgx\_unseal(sealed\_gk)$
5 $y_{add} \leftarrow E_{h(b_{add})}(\texttt{gk})$
`end ecall`
6 Store: $\langle \{u_{add}\}, y_{add}, c_{add} \rangle$
7 **else**
8 $p_{add} \leftarrow RandomItem(\mathcal{P}')$
9 $p_{add} \leftarrow p_u \cup \{u_{add}\}$
`begin ecall`
10 $c_{add} \leftarrow sgx\_add\_user\_to\_partition(\texttt{MSK}, p_{add}, u_{add})$
`end ecall`
11 Update: $\langle \forall u \in p_{add}, *, c_{add} \rangle$
12 **endif**
13 $\mathcal{P} \leftarrow p_{add} \cup \mathcal{P}$

Alg. 3 CON-SKY 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 $p_{rem} \leftarrow p_{rem} \setminus \{u_{rem}\}$
`begin ecall`
3 $\texttt{gk} \leftarrow$ RandomKey()
4 $(b_{rem}, c_{rem}) \leftarrow sgx\_remove\_user(\texttt{MSK}, p_{rem}, u_{rem})$
5 $y_{rem} \leftarrow E_{h(b_{rem})}(\texttt{gk})$
6 **for** all $p \in \mathcal{P} \setminus p_{rem}$ **do**
7 $(b_p, c_p) \leftarrow sgx\_rekey\_partition(p)$
8 $y_p \leftarrow E_{h(b_p)}(\texttt{gk})$
9 **endfor**
10 $sealed\_gk \leftarrow sgx\_seal(\texttt{gk})$
`end ecall`
11 Update: (1) $\langle \forall u_i \in p_{rem}, y_{rem}, c_{rem} \rangle$ (2) $\forall p \in \mathcal{P} \setminus p_{rem} : \langle *, y_p, c_p \rangle$

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.

Re-partitioning

As many removal operations can result in partially unoccupied partitions, we propose the use of a re-partitioning scheme whenever the partition occupancy is 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.

CON-SKY Implementation

In order to implement the system, we used the PBC [Lynn et al., 2006] *pairing-based cryptography* library which, in turn, depends on GMP [Granlund et al., 1991] to perform arbitrary precision arithmetic. They both have to be used inside SGX enclaves. There are several challenges when porting legacy code to run inside enclaves. Besides having severe memory limitations (Section 2.2), 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 run-time, 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 [Intel, 2017a], its Advanced Encryption Standard (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 [Intel, 2017b]. 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.

4.3 Security Analysis

Our reductionist security analysis proposes arguing the security of IBBE-SGX by reducing it to three underlying hard problems : the robustness of TEE, the security of formal IBBE-SGX specification with no partitioning (see Section 4.1.3), and finally the robustness of the partitions mechanism. We infer that adversaries can not break our cryptographic system unless they can break one of the three hard problems aforementioned.

Robustness of TEE

TEE robustness is relying on the assumption that Intel is trusted for implementing the hardware module and software SDK of the SGX technology. Moreover, we state as orthogonal any side-channel vulnerability [Lee et al., 2017] and consider that such issues will be addressed by future iterations of the technology. Both the hardware enclave and the code running inside Intel SGX are attested as genuine before being provisioned with long term secrets (i.e., the master secret key M_{SK}). For a comprehensive security model and definitions of TEE as a *secure hardware scheme*, the reader is referred to *Fisch et al.* [Fisch et al., 2017].

Security of IBBE-SGX without partitions

We show that IBBE-SGX without the partitioning mechanism – formalized in Section 4.1.3 – is equivalent to the traditional IBBE. Traditional IBBE has been proved to achieve *semantic security* [Delerablée et al., 2007]. In doing so, the authors provided a computational security proof that makes use of a generalization of the Diffie-Hellman exponent assumption – as a hard problem – within the context of pairing based cryptography [Boneh et al., 2005a].

By accepting the hardness of TEE, the *semantic security* of traditional IBBE, and the output equivalence of IBBE-SGX with IBBE, then the *semantic security* of IBBE-SGX transitively follows. We will first show the correctness of IBBE-SGX in Theorem 1 and then it's equivalence to IBBE in Theorem 2.

Theorem 1. *IBBE-SGX satisfies* ***correctness*** *as :* $\mathtt{gk}$ *= De-envelope(*u_{SK}*, Envelope(*M_{SK}*,* PK*)).*

Proof. Let β be the base and p the exponent of the de-enveloping Formula 4.7, such that $\mathtt{gk} = \beta^p$. Then by using the same Formula 4.7 :

$$\beta = e(C_1, h^{\mathcal{P}}) \cdot e(U_{sk_r}, C_2) = e(w^{-k}, h^{\mathcal{P}}) \cdot e\Big(g^{(\gamma+\mathcal{H}(u))^{-1}}, h^{k\cdot\prod_{u\in S}(\gamma+\mathcal{H}(u))}\Big)$$

$$= e(w,h)^{-k\mathcal{P}} \cdot e(w,h)^{k\prod_{u\in S\setminus\{r\}}(\gamma+\mathcal{H}(u))} = e(w,h)^{k\prod_{u\in S\setminus\{r\}}\mathcal{H}(u)} = \mathtt{gk}^{\frac{1}{p}}$$

$$\beta^p = \mathtt{gk}^{p^{-1}p} = \mathtt{gk}$$

□

Theorem 2. *Given an identical set of values $\pi = (g, h, \gamma, k)$, the two schemes IBBE(π) and IBBE-SGX(π) produce identical* gk *and envelope* $\mathcal{C} = (C_1, C_2, C_3)$.

Proof. gk, C_1, and C_3 are computed by identical formulas for the two schemes, concretely Formulas 4.2, 4.3, 4.6. Differently C_2 is computed by Formula 4.5 for IBBE and Formula 4.4 for IBBE-SGX. The equivalence of the two formulas is immediate as the second one makes use of a polynomial expansion of the first one. □

Robustness of partitioning

The partitioning mechanism makes use of symmetric encryption to encapsulate the overall group key gk using the partition key – the latter obtained through IBBE-SGX. Symmetric encryption achieves *semantic security* by using a randomization that ensures non-determinism for the obtained chipertext – addressed by a sufficiently large (e.g., 256 bits) initialization vector used only once and an encryption mode that propagates the non-determinism (e.g., chaining mode). Such constraints are included in the system's implementation.

4.4 Closing Remarks

Within this chapter we have presented CON-SKY: an end-to-end group sharing system with confidentiality guarantees. CON-SKY uses a change of assumption of traditional Identity-Based Broadcast Encryption (IBBE) schemes by requiring the enveloping operation to be run by a Trusted Authority (TA) in SGX. CON-SKY also provides a partitioning design that can efficientize the performance at the user side.

A number of further improvements can be argued for CON-SKY. 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 CON-SKY to a distributed set of administrators that would perform membership changes concurrently on the same group or partition by using lock-free techniques. Moreover, in a setup with multiple administrators, one can envision certifying blocks of membership operations logs through blockchain-like technologies.

Even though CON-SKY's underlying cryptographic scheme (i.e., IBBE-SGX) achieves *semantic security*, the system does not consider the privacy of its participants but only the confidentiality of the shared data. As such, IBBE-SGX similarly to IBBE does not provide *anonymity* guarantees. A user always knows the peer members in the group. However, in more constrained threat models, one might desire that participants do not know who else can access the shared encrypted data. Building an *anonymous* key distribution mechanism requires the investigation of other cryptographic constructs than IBBE or IBBE-SGX. The following Chapter 5 provides a novel cryptographic construct and an end-to-end system

ANO-SKY capable of *anonymity* guarantees, that like CON-SKY leverages TEE for a minimal set of operations.

CHAPTER

5

ANO-SKY: Anonymous Key Distribution with TEE

State-of-the-art *anonymous* key management is inefficient when considering large and highly dynamic group membership. Within this chapter we describe ANO-SKY, an innovative anonymous data sharing system that leverages Trusted Execution Environments (TEE) for key distribution and *writing* data to the cloud. Moreover, ANO-SKY uses an indexing mechanism that leverages TEE to optimize the user computational time.

We describe ANO-SKY 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.

The novel construct ANO-SKY has been published in the *International Symposium on Reliable Distributed Systems* (SRDS) [Contiu et al., 2019b].

Within *end-to-end security*, the data shared by a group is encrypted with a secret group key `gk`. Broadcasting the key to *active* group users is done through a key management mechanism. Besides *confidentiality*, this chapter focuses on the *anonymity* guarantee of such key management mechanism in which *active* users should not be able to infer who else is an *active* member of the group. 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 [Dwyer III et al., 2004] and therefore fellow patients cannot infer their identity.

State-of-the-art *anonymous* key management relies on *hybrid encryption* - a symmetric encryption of the message by `gk` and the construction of a key envelope by asymmetric encryptions of `gk`. To ensure *anonymity*, no mapping metadata is transmitted to recipients (Section 2.5). As such each recipient performs trials of asymmetric decryptions to find the

symmetric key within the key envelope. We already argued *hybrid encryption* as inefficient due to the computational cost of public key operations, and also due to the size expansion of the key envelope (Figure 2.3b).

Adapting IBBE-SGX, our contribution from the previous CON-SKY Chapter 4, to support *anonymity* is not straightforward. IBBE-SGX requires the group composition to be known at de-enveloping time (see Equation 4.7 in Section 4.1.3), thus breaking anonymity. In addition, to prevent *active* attackers to reuse `gk` by broadcasting malicious crafted messages [Barth et al., 2006], it is imperative that `gk` changes per each message. Therefore the partitioning mechanism of IBBE-SGX in which only partitions sub-keys change but not `gk` is not applicable in the *anonymous* setting.

Our solution ANO-SKY minimally makes use of Trusted Execution Environments (TEE) and conceptually relies on two paradigms : a cryptographic key management solution and a data delivery protocol. By handling keys within TEE the enveloping can be performed without leaking users identities. Moreover, TEE enables the replacement of the costly public key primitives with symmetric ones, enabling ANO-SKY to be practical efficient.

5.1 Key Management in ANO-SKY

ANO-SKY leverages Intel Software Guard Extensions (SGX) as a TEE. In order to avoid passing all the system operations through a TEE-enabled *administrative 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 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 [Barth et al., 2006; Libert et al., 2012] can be modified to accommodate a new entity of trust for the key enveloping operation, therefore allowing more efficient operations.

Figure 5.1 displays the overview of ANO-SKY solution. The ANO-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.

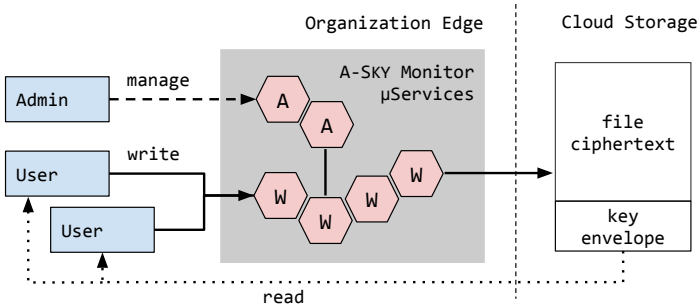


Figure 5.1 – ANO-SKY solution overview. ANO-SKY monitor services are ACCESSCONTROL (**A**) and WRITERSHIELD (**W**).

Key Management Change of Assumption.

The first building block of ANO-SKY 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 [Barth et al., 2006], 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 [Barth et al., 2006] 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.

ANO-SKY allows users to write the encrypted shared content through the WRITERSHIELD service, which acts as a proxy. The service checks 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.

5.2 TEE Trust Establishment

Before relying on any service of the ANO-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* (see Section 2.3).

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* [Costan and Devadas, 2016]) 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 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 [Costan and Devadas, 2016]. 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.

5.3 Operations Design

This subsection formally defines the operations of the ACCESSCONTROL and WRITERSHIELD services.

We recall from Section 2.1 that $E_k(p) \rightarrow c$ and $D_k(c) \rightarrow p$ define symmetric encryption and decryption algorithms; $AE_k(p) \rightarrow (c,t)$ and $AD_k(c,t) \rightarrow \{p, \perp\}$ authenticated encryption and decryption algorithms; $S_{pri}(p) \rightarrow \sigma$ and $V_{pub}(p,\sigma) \rightarrow \{true, \perp\}$ digital signature and verification schemes; h denotes a one way cryptographic function and $||$ denotes the literal concatenation operation.

ACCESSCONTROL Operations.

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 (Section 5.2).

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.

Alg. 4 Ano-Sky Key enveloping (AccessControl)

Input : user identity u_{id}, group identifier g_{id}, symmetric key `gk`.
Output : an $envelope$ ciphertext of the access control key.

```
   begin ecall
1      envelope ← ∅
2      if u_id ∈ group^w[g_id] then
3          for all users u ∈ group^r[g_id] do
4              u_sk ← keys[u]
5              (c_k, t) ← AE_{u_sk}(gk)
6              envelope ← envelope ∪ {(c_k, t)}
7          endfor
8      endif
9      Return envelope
   end ecall
```

The operation of enveloping an access key for a group of anonymous members is denoted by *KeyEnveloping* and is depicted in Alg. 4. Given the identity of the writing user, the group unique identifier and `gk`, 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 `gk` using the secret key of each group member (lines 3-7).

WriterShield Operations.

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. 5). 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

Alg. 5 ANO-SKY Proxy file (WRITERSHIELD)

Input : user identity u_{id}, group identifier g_{id}, file ciphertext $\mathcal{C}$, ACCESSCONTROL instance $\mathcal{A}$.

```
   begin ecall
1      if u_id ∈ group^w[g_id] then
2          σ ← S_sign_TA(C)
3          Upload to cloud : (C, σ)
4      endif
   end ecall
```

TA signature (line 2). Both parts, file ciphertext and the corresponding signature, are finally uploaded to the cloud (line 3).

Alg. 6 ANO-SKY 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   gk ← Random symmetric key
2   envelope ← A.KeyEnveloping(u_id, g_id, gk) i.e., Alg. 4
3   cipher ← E_gk(P)
4   C ← envelope || cipher
5   W.ProxyFile(u_id, g_id, C, A) i.e., Alg. 5
```

User Operations.

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. 6, 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. 7. As previously stated, reading operations do not involve services running in a TEE. The first step is to download the 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

Alg. 7 ANO-SKY User read file

Input : user secret key u_{sk}.

```
1   Download from cloud: (𝒞, σ)
2   if V_{pub-sign_TA}(𝒞, σ) ≠ ⊥ then
3       envelope, cipher ← split(𝒞)
4       for all pairs (c_k, t) in envelope do
5           gk ← AD_{u_sk}(c_k, t)
6           if gk ≠ ⊥ then
7               P ← D_gk(cipher)
8               Return P
9           endif
10      endfor
11  endif
12  Return ⊥
```

by using the user secret key u_{sk} (lines 4-5). If successful, the obtained plaintext is the gk, that the user can use to symmetrically decrypt the file ciphertext (lines 7-8).

5.4 Indexing for Efficient Decryption

Following the methodology of traditional anonymous broadcast encryption (ANOBE) schemes [Barth et al., 2006; Libert et al., 2012], 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. 7) 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 [Barth et al., 2006] or by using the theoretical constructs of tag-based encryption [Libert et al., 2012]. Given the change of assumption brought by ANO-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. ANO-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. 8) 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).

Alg. 8 ANO-SKY Key enveloping with *efficient decryption.*

Input : user identity u_{id}, group identifier g_{id}, symmetric key $\mathtt{gk}$.
Output : an $envelope$ ciphertext of the access control key.

```
1   envelope ← ∅
2   if u_id ∈ group^w[g_id] then
3       nonce ← Random
4       for all users u ∈ group^r[g_id] do
5           u_sk ← keys[u]
6           l_u ← h(u_sk || nonce)
7           (c_k, t) ← AE_{u_sk}(gk)
8           envelope ← envelope ∪ {(l_u, c_k, t)}
9       endfor
10      Sort envelope by l (i.e.,, label)
11  endif
12  Return nonce || envelope
```

Alg. 9 ANO-SKY User read file with *efficient decryption.*

Input : user secret key u_{sk}.
Output : file plaintext.

```
1   Download from cloud : (C, σ)
2   if V_{pub-sign_TA}(C, σ) ≠ ⊥ then
3       nonce, envelope, cipher ← split(C)
4       l_u ← h(u_sk || nonce)
5       (c_k, t) ← binary search for key : l_u in envelope
6       if (c_k, t) ≠ ⊥ then
7           gk ← AD_{u_sk}(c_k, t)
8           P ← D_{gk}(cipher)
9           Return P
10      endif
11  endif
12  Return ⊥
```

A user read operation (Alg. 9) 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 $\mathtt{gk}$ 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

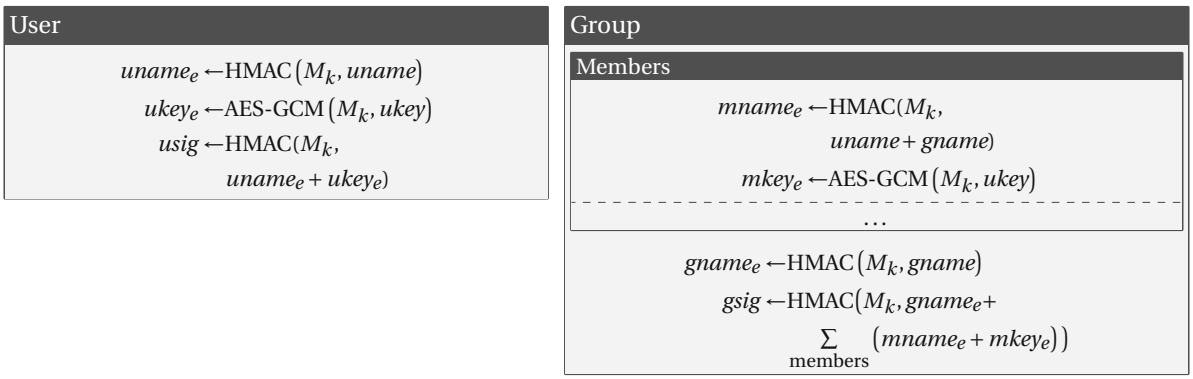


Figure 5.2 – Data model of user and group *documents* stored in MongoDB.

is reflected during decryption time, replacing $O(n)$ trials of symmetric decryption with a $O(\log n)$ binary search and a single symmetric decryption.

5.5 Implementation

AccessControl

The AccessControl service is the only stateful component of Ano-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 sub-components. The first one constitutes the entry-point for service requests. It is developed in C++, for a total of 3000 LoCs.

The other one holds users and groups metadata within a replicated database. For this purpose, we use a triple-replicated cluster of MongoDB [MongoDB Inc., 2019] 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 [MongoDB Inc., 2018] 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 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. 5.2 shows how we organize data 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 denormal-

ization 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 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 [Lohmann et al., 2018]. In order to terminate TLS connections in the enclave, we use a port of OpenSSL for SGX [Intel, 2017b].

Another duty of the ACCESSCONTROL service is to generate key envelopes upon user requests. An envelope contains a `gk` encrypted several times, once per group member. The `gk` as well as the user keys, are 32 bytes long. We use AES GCM, which generates a *tag* of 16 bytes for integrity. Considering the addition of 12 bytes for the initialization vector, each group member adds 60 bytes to the envelope.

In order to avoid having to perform $O(n)$ decryption trials, we can index the keys within the envelope (Section 5.4). 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 bytes 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.

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-CCA). 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 [Pires et al., 2018]. 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 [Minio, Inc., 2019], 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++. Without accounting for external libraries, the WRITERSHIELD consists of 800 LoCs.

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 ANO-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) [Shipilev et al., 2018] and Yahoo! cloud serving benchmark (YCSB) [Cooper et al., 2010] frameworks that we use to perform the evaluation of ANO-SKY (Section 7.2.2).

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 [Vaucher et al., 2018]. 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.

5.6 Security Analysis

We discuss the security guarantees of ANO-SKY and provide a brief intuition for the formalism of a reductionist security proof. We hypothesize that ANO-SKY achieves indistinguishability with respect to adaptively-chosen ciphertexts (i.e., IND-CCA2). 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 ANO-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 or *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 ANO-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 [Bellare and Namprempre, 2000]. Such a guarantee forces to choose a specific AE mode for AES, as described in Section 5.5. On the other side, TEEs have been used in the composition of functional encryption cryptographic primitives shown to achieve IND-CCA2 guarantees [Fisch et al., 2017]. In the following, we retain the formalism of Fisch *et. al.* [Fisch et al., 2017] 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 ANO-SKY is IND-CCA2.

Proof. We provide a reductionist method that lays the frame for a formal proof. ANO-SKY can be seen as a reduction of the anonymous broadcast encryption scheme of Barth *et. al.* [Barth et al., 2006] (*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 [Libert et al., 2012]), relying on the two aforementioned replacements, one can construct identical adversary-challenger game steps (Def. 2 in [Libert et al., 2012]) and employ a similar sequence of experiments (Appendix A in [Libert et al., 2012]) that can prove that ANO-SKY is immune to chosen ciphertext attacks.

5.7 Closing Remarks

We have presented ANO-SKY, an anonymous file sharing end-to-end system. ANO-SKY achieves practical efficiency by performing the key management within trusted execution environments (TEE) and for *writing* the data to the cloud. A comprehensive performance benchmark of ANO-SKY is presented in Chapter 7.

Future work could enable ANO-SKY to benefit from *edge* computing gateways sitting at the border of the organization. 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.

Even tough ANO-SKY has a small key envelope – 60 bytes per group member – it is unsure if an *anonymous* key management scheme can achieve both small *constant* envelope size – no matter the group size – and efficient computational time. If for *confidentiality-*

only schemes such construct is possible (Section 4.1.3), for *anonymous* schemes such challenge is an open research problem.

CHAPTER

6 REV-SKY: Practical Revocation with TEE

A full re-encryption during revocation can have prohibitive costs for very large file resources. This chapter presents REV-SKY, a practical active revocation scheme leveraging All-or-Nothing Transform (AONT) and Trusted Execution Environments (TEE). REV-SKY is illustrated by its three main components : (1) The trust establishment protocol attesting the re-encryption workers; (2) The protocol that the users follow to read and write files; (3) The distributed re-encryption protocol leveraging SGX enclaves. Following implementation details, the chapter ends with an analysis of the security guarantees.

End-to-end security specifies that user data is encrypted before sent to the untrusted storage. If the two previous chapters focused on the distribution mechanism of the encryption key – namely `gk` – the current chapter centers on what happens to the data during revocation.

State of the art revocation methods inflict a penalty on security by delaying re-encryption to the first update of the data (e.g., *lazy revocation*) or differently favor security by re-encrypting the entirety of the data (e.g., *active revocation*). Such methods trade-off security for efficiency or the other way round. REV-SKY achieves both efficiency and security by presuming two assumptions : (1) users that downloaded and *viewed* the entirety of a resource (e.g., *file*) can not be prevented to continue access their local copy of the file, and (2) the storage provider is equipped with TEE capabilities.

As such, REV-SKY leverages TEE for data locality performing re-encryption directly on the storage provider side. Only portions of the data (i.e., *super blocks*) need to be re-encrypted by REV-SKY due to an All or Nothing Transform (see Section 2.6) performed on the user side. Moreover, REV-SKY prevents malicious users to provision super blocks by making the latter unknown unless the user downloaded the whole file resource.

Assumptions

REV-SKY uses standard cryptographic primitives, making it both easy to parse and implement. Recall that $E_k(d)$ and $D_k(d)$ denote symmetric encryption and decryption; $h(d)$ a one-way cryptographic function; $AE_{pub}(d)$ and $AD_{pri}(d)$ asymmetric encryption and decryption primitives and $S_{pri}(d)$ and $V_{pub}(d)$ are asymmetric signature and verification. Let $\oplus$ denote *exclusive or* operation.

We assume the existence of past (i.e., `gk`) and new (i.e `gk'`) group keys. The latter is known only to non-revoked members of the group. Moreover, we assume that the storage service provider is equipped with a set of machines with SGX capabilities. Each SGX enclave is equipped with a unique private key `sgx-key`, accessed locally using the quoting enclave [Costan and Devadas, 2016]. Its corresponding public key is retained by Intel.

To better isolate and explain our revocation method, we simplify the model actors to *data owners* that make data available to *users* over untrusted storage. Within this simplified model *data owners* create content while *users* only consume content. *Data owners* are also *administrators*, making `gk` known to active users by a key management technique (e.g., CON-SKY of Chapter 4). Moreover, re-encryption *workers* are computing agents sitting on the cloud storage performing the actual re-encryption tasks within TEE.

6.1 Trust Establishment

The bootstrapping of the system consists of establishing trust among the agent triggering the revocation (i.e., the data owner) and the re-encryption workers that enable it. Re-encryption workers need to certify the validity of the administrator, while the administrator must certify that the workers have SGX capabilities and are instantiated with the REV-SKY code. Attested SGX enclaves are provided with a long term REV-SKY private key (i.e., `sgx-key`). This key will be subsequently used by the re-encryption protocol. Trust can be established with an arbitrary number of storage-side re-encryption workers, that we simply denote to as *workers* in the following.

Figure 6.1 illustrates the methodology for trust establishment. A higher trusted certifying authority (CA) is being used as a point of trust for both administrators and workers.

In a first step, the administrator authenticates its public key, signed by the CA. Upon the successful verification by using the CA's public key, workers retain the administrator public key with the scope of accepting re-encryption requests exclusively signed by this administrator key.

In a second step, each worker produces two items : (1) a digest of the code and data instantiated in the SGX enclave (commonly refereed to as a *quote* in the SGX literature) and (2) the public component of an asymmetric key generated inside the enclave. The worker signs both the digest and the enclave public key using the fused SGX key `sgx-key`.

In the third step, the administrator contacts the Intel Attestation Service and validates that the signature of the quote is indeed a genuine SGX key.

In the final step, the administrator provides to the enclave the long term `rsky-key`. To do so, the administrator encrypts `rsky-key` using the public key sent by the enclave, and signs it using the administrative private key. Upon receiving the ciphertext, the workers decipher `rsky-key` after validating the administrator signature. To persist the `rsky-key`, enclaves use the standard sealing mechanism offered by SGX [Costan and Devadas, 2016] (see Section 2.2).

6.2 Operations Design

Client Write with REV-SKY

In the following we describe the protocol that the data owner follows to write a file to the untrusted storage, such that REV-SKY can be used to implement active revocation operations.

The core of the solution is to employ an All or Nothing Transform together the super-encryption of some of the resulted blocks by using the group key `gk`.

Let $\mathcal{D}$ represent an input data file. The write operations proceeds by splitting $\mathcal{D}$ in equal size blocks $d_0, d_1, ..., d_n$. An AONT follows, by generating a random symmetric key, that we refer to as the File Key FK. This key is used for encrypting each block of $\mathcal{D}$:

$$(c_0,\ c_1,\ ...\ c_n) = (E_{FK}(d_0),\ E_{FK}(d_1),\ ...\ ,\ E_{FK}(d_n)) \tag{6.1}$$

The hashes of the resulted ciphertexts are then chained together with FK using successive exclusive-or operation, forming a tail packet, that we refer to as metadata:

$$meta_{FK} = h(c_0) \oplus h(c_1) \oplus ... \oplus h(c_n) \oplus FK \tag{6.2}$$

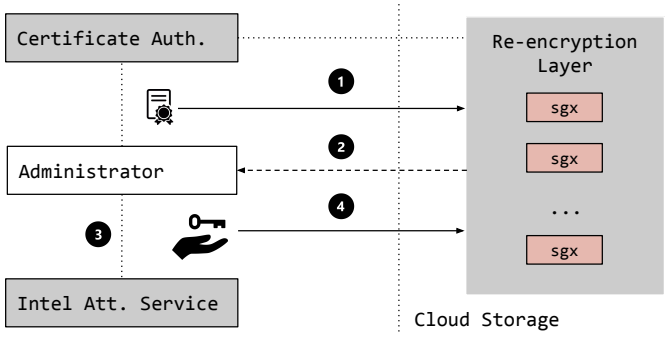


Figure 6.1 – REV-SKY trust establishment protocol.

One can notice that possessing all ciphertexts $(c_0, ..., c_n)$ and $meta_{FK}$ allows deriving FK. In order to prevent the finding of the entire set of hashes unless `gk` is known, we randomly choose a subset of ciphertext blocks, and super encrypt them by using `gk`. We call these blocks the *super blocks*:

$$(se_i, ..., se_k) = (E_{\texttt{gk}}(c_i), ..., E_{\texttt{gk}}(c_k)) \tag{6.3}$$

Non-super blocks are using the single encrypted version obtained in Equation 6.1. We push to the cloud storage the super blocks, the metadata packet, and the non-super blocks. When a re-keying happens, only the super blocks need to be re-encrypted from `gk` to `gk'`.

Our scheme is so far secure against malicious users who are not in the possession of `gk`. However, knowing the indices of super blocks ($i, ..., k$ in Equation 6.3) prior to downloading a file allow easily implementing a pre-provisioning attack by selectively downloading a version of these blocks and use this stored version to access the file after their re-encryption, by deriving FK. To protect against such attack, and make sure that the power of malicious users is not greater than with full-file re-encryption, we constrain the active users to download an entire file before being able to find out which of the file's blocks actually are super blocks. We employ for this purpose a *second* AONT that allows only revealing the indices of the super blocks once the whole file has been downloaded, but not before. Let SK be a symmetric key, used for encrypting the indices of the super blocks:

$$meta_{index} = E_{SK}(i, ..., k) \tag{6.4}$$

The second AONT xor-chains SK together with the hashes of the blocks that were stored in the cloud:

$$meta_{SK} = h(c_0) \oplus ... \oplus h(se_i) \oplus ... \oplus h(se_k) \oplus ... \oplus h(c_n) \oplus SK. \tag{6.5}$$

To allow access to $meta_{SK}$ only by valid group members, we over-encrypt it using `gk`.

Only the super blocks are subject to re-encryption when `gk` changes to `gk'`. This means that a re-encryption worker needs to know the indices of these super blocks. Rather than requiring a worker to perform the reverse AONT, requiring a processing cost linear in the file size, we use a fact-track, constant-cost alternative. We ask that during the REV-SKY write operation, the key SK that reveals the indices be encrypted by the public key associated with `rsky-key`. Re-encryption workers are in the possession of the corresponding decryption key, provided at the end of trust establishing protocol (Subsection 6.1). Therefore, a new metadata package is constructed:

$$meta_{sgx} = AE_{\texttt{rsky-pub-key}}(SK) \tag{6.6}$$

Finally, the metadata packages are appended to the stored content on the cloud. Figure 6.2 illustrates the complete REV-SKY write procedure with two super blocks for a file. Steps 1 to 3 correspond to the first AONT. Step 4 performs the super encryption of two blocks. Step 5 encrypts the super block index, and step 6 implements the second AONT. Step 7 asymmetrically encrypts the access to super block indices, while step 8 lists the packages that are copied to the cloud.

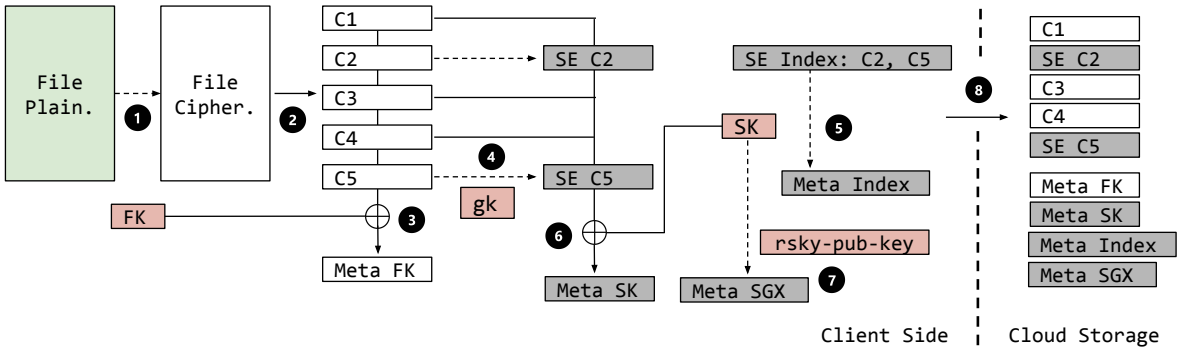


Figure 6.2 – REV-SKY client writing a file to the storage.

Client Read with REV-SKY

We describe the protocol a user needs to follow for reading a file from the cloud storage. We consider that this user is a legitimate group member in possession of `gk`. We assume that the data owner provides the user with the list of files and corresponding blocks when registering it to a dataset group.

Once all the blocks for a file are downloaded from the cloud, the user decrypts $meta_{SK}$ by using `gk`. Next, the client performs the reverse of the second AONT in order to find SK, the key that allows the decryption of the indices of super blocks. Upon knowing the indices of super blocks, the client can decipher them by using `gk`. The super blocks plaintext is used with the rest of the blocks for the reverse of the first AONT, which reveals FK and allows decrypting the file plaintext.

REV-SKY Re-encryption Protocol

Active revocation can take place as soon as re-encryption workers are validated and provisioned with `rsky-key`, which they can use for determining and accessing super-blocks. Re-encryption is performed in parallel over a number of workers. As the number of workers increases, the probability of a crash of one of the workers also increases. The distributed re-encryption must therefore tolerate faults and ensure the completeness of the operation, i.e., that all files' super blocks have been properly re-encrypted with the new key. REV-SKY builds on a replicated coordination kernel (Apache ZooKeeper in our implementation) to assign and monitor the completion of the distributed re-encryption tasks.

The re-encryption protocol is triggered by the data owner (i.e., *administrator*), who starts by splitting the list of files shared by the group into batches, forming re-encryption tasks. These tasks are orchestrated by a fault-tolerant master process built over the coordination kernel. Each task is associated with metadata, including the values of keys `gk` and `gk'` encrypted with the provisioned REV-SKY key, only allowing the attested enclaves to

access plaintext keys:

$$meta_{task} = S_{admin-key}(AE_{\texttt{rsky-pub-key}}(\texttt{gk}, \texttt{gk'})) \tag{6.7}$$

Next, workers receive the task metadata from the master. They check its authenticity, i.e., that it has been created by an administrator, by validating its signature. They decrypt keys `gk` and `gk'` as

$$\texttt{gk}, \texttt{gk'} = AD_{\texttt{rsky-key}}(V_{admin-key}(meta_{task})) \tag{6.8}$$

At this point the worker starts processing the task. For each file in the task, the worker fetches the index metadata ($meta_{sgx}$ and $meta_{index}$).

In order to determine the indexes of the super blocks, it calculates SK (Equation 6.9) and uses it to symmetrically decrypt the index (Equation 6.10):

$$SK = AD_{\texttt{rsky-key}}(meta_{sgx}) \tag{6.9}$$

$$(i, ..., j) = D_{SK}(meta_{index}) \tag{6.10}$$

Knowing the super blocks identity, a worker fetches these blocks for processing within the enclave. It decrypts each super-block by using the old `gk` (Equation 6.12) and encrypts it again using the new key `gk'` (Equation 6.12):

$$(c_i, ..., c_k) = (D_{\texttt{gk}}(se_i), ..., D_{\texttt{gk}}(se_k)) \tag{6.11}$$

$$(se_i, ..., se_k) = (E_{\texttt{gk'}}(c_i), ..., E_{\texttt{gk'}}(c_k)) \tag{6.12}$$

Moreover, as the hashes of super blocks are changed, $meta_{SK}$ needs to be updated, so that the clients can perform an AONT to find SK. Therefore, $meta_{SK}$ is downloaded from the storage to the enclave, and decrypted by using `gk`. Old digests are hashed out from $meta_{SK}$ while the new ones are hashed in, by executing Equation 6.13 twice, before and after the super blocks re-encryption. Finally, $meta_{SK}$ is encrypted by `gk'` and pushed to the storage:

$$meta_{SK} = meta_{SK} \oplus h(se_i) \oplus ... \oplus h(se_k) \tag{6.13}$$

The worker finally pushes back the re-encrypted super blocks to the storage. It then signals task completion by notifying the master using the coordination kernel.

The master process monitors the progress of tasks by the workers, and can re-assign tasks of failed workers to alive ones if necessary. This may result in some duplicated work but does not bear risks of data corruption. A failure of the master is mitigated by the spawn of a new master, who recovers from the state stored in the replicated coordination kernel – this follows the classical master-worker task distribution described by Junqueira and Reed [Junqueira and Reed, 2013].

We note that a balance of the load between workers is easier to obtain than with full re-encryption. The revocation requires re-encryption of a fixed number of super blocks

per file, and the super block identifiers are available to the enclave. This results in a fixed cost per file, regardless of its size, allowing balancing the load by balancing the number of files in the assigned tasks. Differences in execution speed between workers and resulting imbalances could be addressed by implementing a form of work stealing, which we leave to future work.

Size and Number of Super Blocks

The number of super blocks, and the fixed size of all blocks, are parameters that can be set by the data owner.

As we detail in the following security analysis Section 6.4, a single super block is actually sufficient if we consider the attacker's benefit expectation in terms of accessible files after revocation. A malicious user implementing a pre-provisioning attack cannot gain access to more files in expectation than a malicious user downloading, decrypting and saving complete files for later use. Using more super blocks only increases the difference between these two strategies, in favor of the latter. However, using more super blocks is a way to increase the cost of the pre-provisioning attacks, requiring more storage space per pre-provisioned file. Using more super blocks increases the cost of the re-encryption at the server side, and marginally increases the cost of the decryption at the client side. Our recommendation is that in the majority of cases, a single super block should be sufficient.

The size of the blocks is a compromise between the costs imposed at the cloud storage side (and, ultimately, on the data owner paying for virtual machines), and the performance observed by the clients when accessing the store. A small block size reduces the cost of re-encryption on the server side, but also results in more I/O operations and calls to the cloud storage for users, which often limits bandwidth efficiency.

6.3 Implementation

Our system architecture (see Figure 6.3) relies on Apache ZooKeeper to coordinate the distributed re-encryption and on Apache Cassandra for the cloud storage. The clients are considered to communicate directly with the storage cluster, while the administrator communicates with the ZooKeeper service.

The Cassandra (v3.11.3) ring stores both files metadata and blocks in a structured manner in two tables linked through a foreign key. The cluster is deployed over 4 physical machines. It is configured to use one replica per node and two seed nodes. The data directory of each Cassandra node points to a folder on the local SSD drive, with an available capability of 2 TB.

Our ZooKeeper (v3.4.12) deployment relies on a single master node. A production deployment would employ several replicated nodes. The distributed master-slave orchestration uses a hierarchy of four types of ZooKeeper *znodes*. The `workers` znodes store the

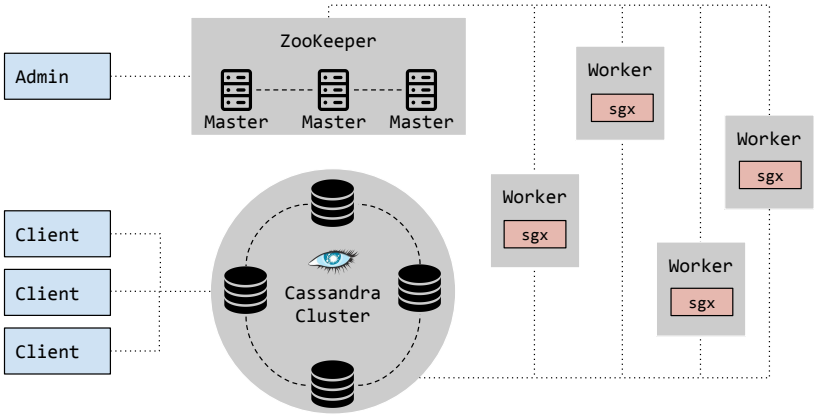


Figure 6.3 – REV-SKY System Architecture.

identifiers of workers that have an active connection to the master. The `tasks` znodes store the re-encryption batches created by the administrator. The `assign` znodes are filled by the master and contain the assignment of re-encryption batches for each worker. Finally, the `status` znodes are filled by re-encryption workers signaling the completion of a batch. The administrator watches these last znodes and decides of a revocation termination when all initial tasks batches have positive termination status. To tolerate failures, the implementation relies on ZooKeeper exceptions in combination with notifications of state changes signaling failure codes.

We utilize AES-256 as our symmetric encryption operation, SHA-256 as the one way function used for the All or Nothing Transform, and asymmetric RSA encryption with OAEP padding with keys of 4,096 bits.

All machines hosting re-encryption workers have CPUs supporting Intel SGX. The re-encryption operation utilizes a single *ecall* while four *ocalls* are utilized for fetching and writing file metadata and blocks. We leverage the `user-check` attribute for data blocks that pass the enclave border, as they are already in ciphertext form. This optimizes enclave transitioning time. We use the `sgx-ssl` library to perform cryptographic operations within an enclave, while the clients rely on `openssl` library. The workers make use of the asynchronous version of C bindings for ZooKeeper.

6.4 Security Analysis

A malicious user implementing a pre-provisioning attack will seek to maximize its gain, defined as the number of files it will have access to past its revocation. The baseline strategy is the download of entire files to local storage during the malicious user's membership

period. To be of interest, the pre-provisioning attack must allow accessing more files after revocation for the same bandwidth budget.

Metadata packages are small in size (a few Bytes). We consider that these are all pre-provisioned by the malicious user. The interest of the attacker is therefore to pre-provision the super blocks prior to their re-encryption.

It is not possible to obtain the indices of the super blocks without an access to all blocks, thanks to the second All-or-nothing Transform. The only possible strategy is to randomly download a subset of the blocks, in the hope that these will contain all the super blocks and allow accessing the file past revocation by downloading missing regular blocks. We analyze how much blocks of data an attacker needs to pre-provision on average in order to fetch the super blocks.

Theorem 3. *Let $s > 0$ be the number of super-encrypted blocks out of a total of n blocks composing a file, with $s \leq n$. Then the average number of block downloads necessary for an attacker to obtain all s super blocks is:*

$$A(n,s) = \frac{s}{s+1}(n+1) \tag{6.14}$$

Proof. We prove by induction. We proceed from the base case $n = 1$, therefore $s = 1$. Clearly, the attacker needs to download the *single* composing block that happens to be a super block (i.e., $A(1,1) = 1$).

We prove next the inductive step for $n+1$, i.e. :

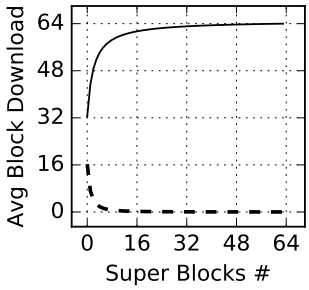
$$A(n+1,s) = \frac{s}{s+1}(n+2) \tag{6.15}$$

When one extra non super block is added, one falls in two outcomes: with probability $\frac{s}{n+1}$ there are $s-1$ out of n blocks left, and differently with probability $\frac{n+1-s}{n+1}$ there are left s out of n blocks. Therefore:

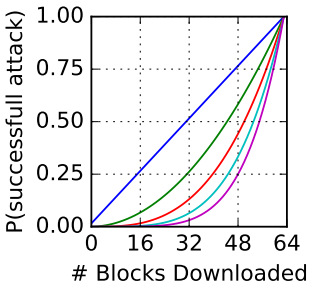
$$\begin{aligned} A(n+1,s) &= \frac{s}{n+1}(1+A(n,s-1)) + \frac{n+1-s}{n+1}(1+A(n,s)) \\ &= 1 + \frac{s}{n+1}\frac{s-1}{s}(n+1) + \frac{n+1-s}{n+1}\frac{s}{s+1}(n+1) \\ &= 1 + (s-1) + \frac{(n+1-s)}{s+1}s = s\frac{n+2}{s+1} = A(n+1,s) \end{aligned}$$

The inductive step for $s+1$ is omitted as it is proved similarly. □

We zoom within the effect of Theorem 3 when using a relatively small number of super blocks. Intuitively, the values of A grow fast for small consecutive values of s (e.g., 1, 2, and 3), as the coefficient of $(n+1)$ goes down from $\frac{1}{2}$ to $\frac{2}{3}$ and to $\frac{3}{4}$. This means that when one super block is utilized the attacker would roughly have to download half of data, for two



(a) The average number of blocks that an attacker has to download (—) and its rate of change (- - -).



(b) The probability that the attacker is successful given a fixed amount of download trials. Super blocks count (left→right): — 1, — 2, — 3, — 4, — 5.

Figure 6.4 – Analysis of a 64 blocks file case for which an attacker tries to provision super blocks.

blocks: two thirds, for three: three fourths and so on. We illustrate this effect in Figure 6.4a using an example of a file composed from 64 blocks. When utilizing just a single super block the attacker needs to download in average 32.5 blocks. With just few increases in the number of super blocks the average number of downloads increases quickly to values approaching 60 out of 64 blocks when utilizing 12 blocks. Therefore, with just few increments in the complexity of the system to put up with, the attacker needs to download and increasingly larger number of blocks to stand a chance of owning useful data comprising all super-blocks. This effect can be concretely observed by illustrating the first derivative $A(n,s)\,dn.$, a rate of change that is inverse proportional to a quadratic number of super blocks (observed with dashed line in Figure 6.4a).

In conclusion, opting for a small number of super blocks can already protect against attackers that would have to inefficiently download large volumes of data. Additionally, small increases when utilizing a small number of super blocks has large implications on the number of blocks an attacker needs to pre-provision.

We generalize now our analysis by considering the total number of block downloads that the attacker is bounded to.

Theorem 4. *Let t be the number of block downloads that an attacker is bound to, n the total number of blocks and s the number of super blocks, where $s \leq t \leq n$. Then the probability that all s blocks were fetched by the attacker is :*

$$P(n,s,t) = \frac{t!\,(n-s)!}{n!\,(t-s)!} \tag{6.16}$$

Proof. Denote by $o = n - s$ the number of ordinary (i.e., not super) blocks. We notice that the probability of discovering all the super blocks is equal to the probability of consecutively discovering only ordinary blocks. Therefore:

$$P(n,s,t) = \frac{n-s}{n} \cdot \frac{n-s-1}{n-1} \cdot \ldots \cdot \frac{n-s-o+1}{n-o+1}$$

$$= \frac{(n-o)!\,(n-s)!}{n!\,(n-s-o)!} = \frac{t!\,(n-s)!}{n!\,(t-s)!}$$

□

Figure 6.4b shows the value of P for the same case of a file of 64 blocks. We consider up to 5 super blocks. This figure actually allows estimating the expectation of the number of accessible files after revocation, for a given fraction of blocks downloaded at random. With a single super block, downloading 50% of the blocks yields an expectation of 0.5 available files. Only with 100% of the blocks can the attacker guarantee an expectation of one available file, falling back to the arguably simpler strategy of downloading entire files in the clear prior to revocation. Adding more super blocks greatly decreases the effectiveness of this partial random download attack: the expectation of accessible files when downloading 75% of the blocks of one file is actually 0.25 file, when using five super blocks. In this case, the strategy of downloading entire files is clearly superior to a pre-provisioning attack, making the latter ineffective.

6.5 Closing Remarks

This chapter introduced REV-SKY, a practical and efficient approach to active data revocation. Active revocation prevents further access to both new and existing data by revoked users. REV-SKY offers active revocation at a fraction of the cost of an implementation using full re-encryption. It splits files in fixed-size blocks and only needs to re-encrypt a fixed small subset of these (typically just one), called super blocks, upon a revocation. The use of All-or-Nothing Transforms effectively prevents pre-provisioning attacks and guarantees the same level of security as full re-encryption. REV-SKY implements efficient and secure re-encryption operations directly at the storage side by leveraging the availability of TEE.

As pointed by our performance evaluation (see Section 7.2.3), REV-SKY can prevent further access to a dataset of 132 TB in only 11 minutes using 5 modest servers, compared to the 8.6 days required for full re-encryption.

Future extension of REV-SKY are envisioned by considering a strengthened threat model than *honest-but-curious* for the cloud storage. For example, the storage can lunch a *rollback attack* by serving outdated content to the re-encryption workers. Such an attack can be mitigated by administrators aggregating proofs of correct data re-encryption or differently by using out-of-the-shelf SGX rollback prevention mechanisms [Matetic et al.,

2017]. Moreover, in a strengthened threat model, one can consider the cloud colluding with malicious users, communicating to them metadata regarding the super blocks. Such a threat can be addressed by enabling re-encryption workers with Oblivious RAM [Stefanov et al., 2013] techniques, already integrated with Intel SGX [Ahmad et al., 2018]. However it is unclear if such a construction provides the necessary efficiency for both data access and re-encryption operations.

CHAPTER

7 Performance Evaluation

This chapter presents a performance evaluation of our three unique contributions CON-SKY, ANO-SKY, and REV-SKY. We first benchmark the functionalities in isolation (*micro*-benchmarks) and then within end-to-end real life scenarios (*macro*-benchmarks).

Results highlight that

— CON-SKY can performs membership changes 1.2 orders of magnitude (OoM) faster than the traditional approach of Hybrid Encryption (HE), producing group metadata that are 3 OoM smaller than HE.

— ANO-SKY cryptographic scheme is 3 OoM better than state-of-the-art ANOBE, and an end-to-end system encapsulating ANO-SKY can elastically scale to support groups of 10,000 users while maintaining processing costs below 1 second.

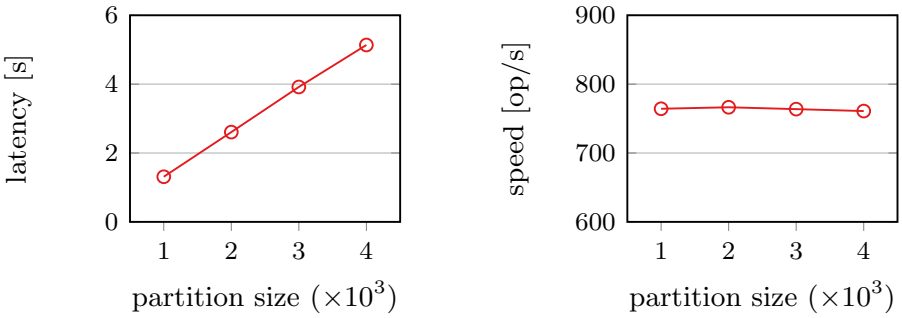
— REV-SKY outperforms active revocation on complete files by up to 3 OoM on existing industry workloads.

7.1 Micro-benchmarks

Micro-benchmarks isolate and measure the performance of specific functionalities. We evaluate each contribution CON-SKY, ANO-SKY, and REV-SKY within a separate dedicated sub-section. The choice of separating the benchmarks is due to the different model and architectural constraints that each contribution implements.

7.1.1 CON-SKY

CON-SKY experiments are 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. We benchmark the



(a) System Set-Up. (b) Extract User Key Throughput.

Figure 7.1 – CON-SKY Bootstrap Performance

CON-SKY system from two different perspectives first by measuring the IBBE-SGX operations performance in isolation, and then by comparing them to Hybrid Encryption (HE). We chose to compare IBBE-SGX only to HE as the latter already shows better computational complexity than IBBE (see Figure 2.3a).

First, we evaluate the performance of the bootstrap phase. This phase consists of setting up the system (Figure 7.1a) and generating secret user keys (Figure 7.1b). 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 7.2 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 Figure 2.3a). Storage-wise, IBBE-SGX is up to 3 orders of magnitude better than HE. Moreover, right column plots of Figure 7.2 zoom 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 kilo-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 7.3a. 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 differ-

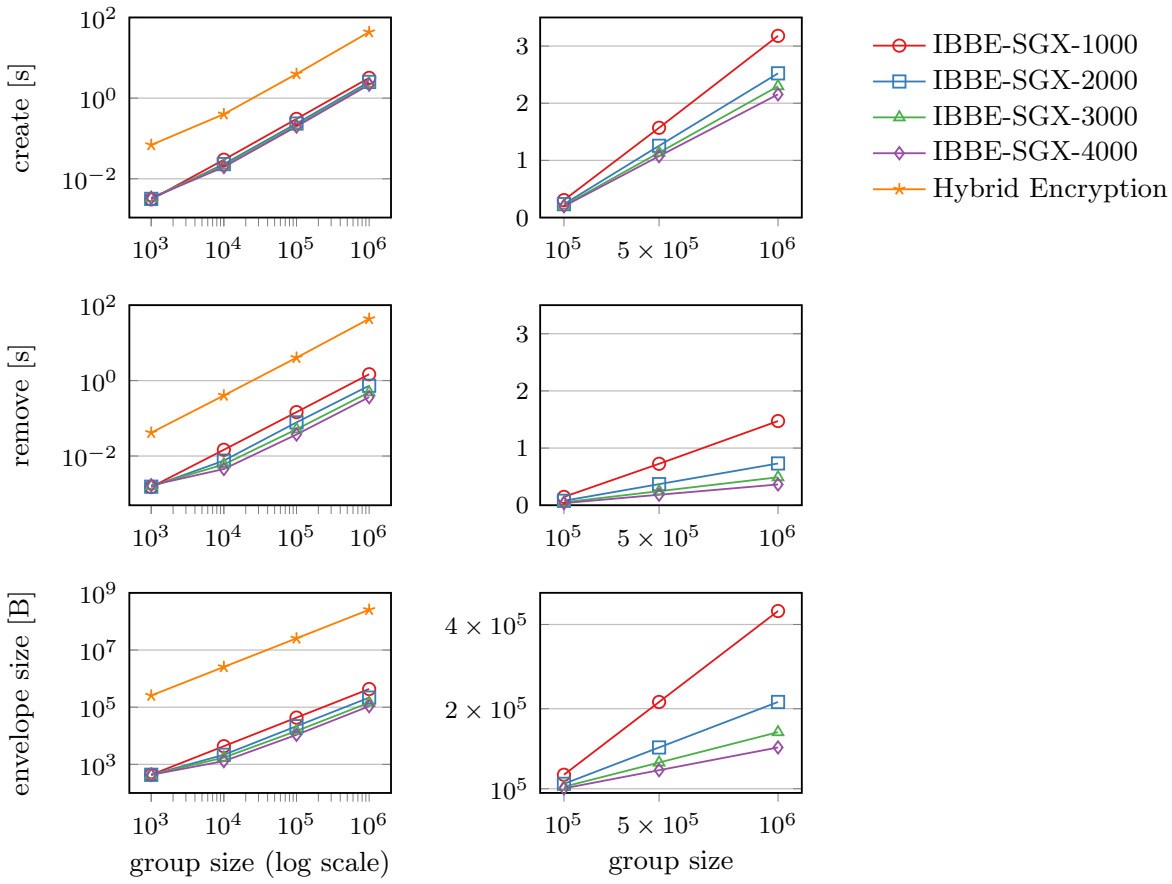
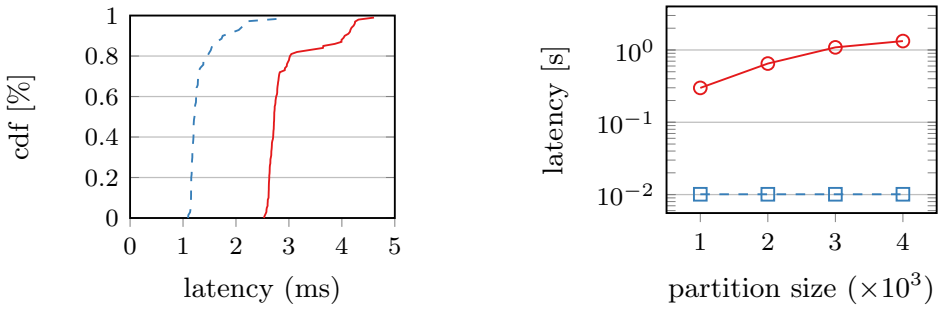


Figure 7.2 – CON-SKY evaluation of create and remove operations and storage footprint, by varying the partition size for IBBE-SGX (1000, 2000, 3000 and 4000).

ence between the two at the CDF value of 0.8. Moreover, the HE add operation is generally twice as fast as IBBE-SGX.

The client de-enveloping performance is shown in Figure 7.3b. The de-enveloping 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 operation. We argue that a slower de-enveloping time for IBBE-SGX can be acceptable in practice. First, the performance is overshadowed by the slow cloud response time necessary for clients to download the group envelope that always precedes a de-enveloping operation. Second, the cost remains bounded to a partition size, independent on the number of users in the group.



(a) Add User to Group Performance.

(b) De-envelope Performance.

Figure 7.3 – CON-SKY performance of the adding a user to a group and de-enveloping operations. Continous red line IBBE-SGX. Dashed line HE.

7.1.2 ANO-SKY

ANO-SKY experiments run on a cluster of 5 SGX-enabled Dell PowerEdge R330 servers, each having an Intel Xeon E3-1270 v6 processor and 64 GB of memory. Additionally, we use 3 Dell PowerEdge R630 dual-socket servers, each equipped with 2 Intel Xeon E5-2683 v4 CPUs and 128 GB 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 [Bulck et al., 2018]. Communication between machines is handled by a Gigabit Ethernet network. When error bars are shown, they represent the 95% confidence interval.

We first isolate and measure the performance of the underlying cryptographic primitive of ANO-SKY. We employ the ANOBE scheme defined by Barth *et. al.* [Barth et al., 2006] (*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 bits of *equivalent security strength* [Barker et al., 2007]. 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 7.1 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, ANO-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 ANO-SKY) primi-

Table 7.1 – Throughput comparison (i.e.,, group size per second: $|\mathcal{G}|$/s) of ANO-SKY cryptographic scheme and *BBW* [Barth et al., 2006], 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	5×10^3	3×10^2	<4
ANO-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*

tives. 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. ANO-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 §6.3, ANO-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*.

We further evaluate the throughput of operations performed by administrators when varying the number of ACCESSCONTROL instances. Requests are distributed among the instances of ACCESSCONTROL by exposing a *service* in Kubernetes. Fig. 7.4 shows our results. The scalability of adding a user to a group or revoking its rights is limited, as these op-

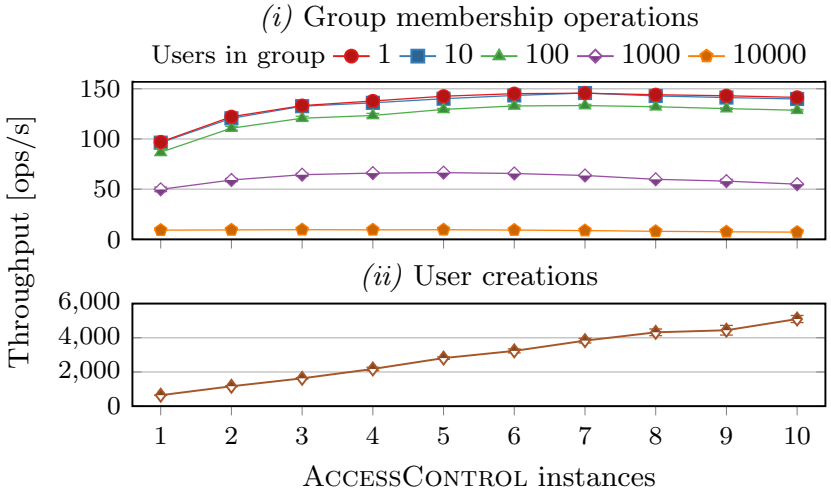


Figure 7.4 – ANO-SKY throughput achieved by ACCESSCONTROL: *(i)* adding or revoking users to/from groups of various sizes, and *(ii)* creating users.

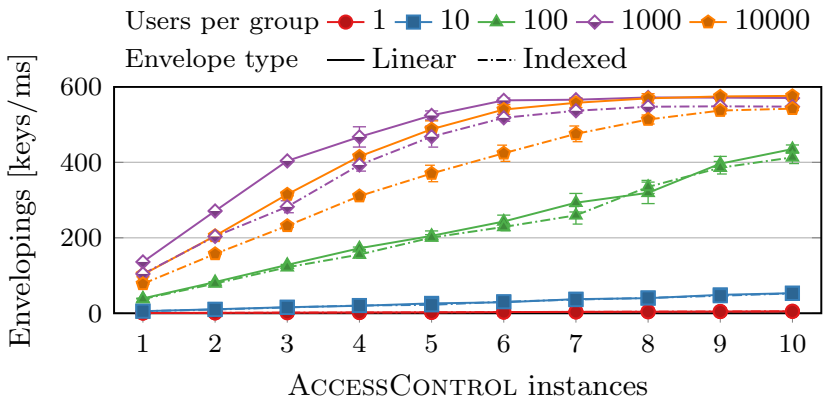


Figure 7.5 – Ano-Sky throughput of enveloping a message for groups of various sizes with varying instances of the AccessControl micro-service.

erations require to perform one 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. 7.5, 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.

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. 7.6, 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-

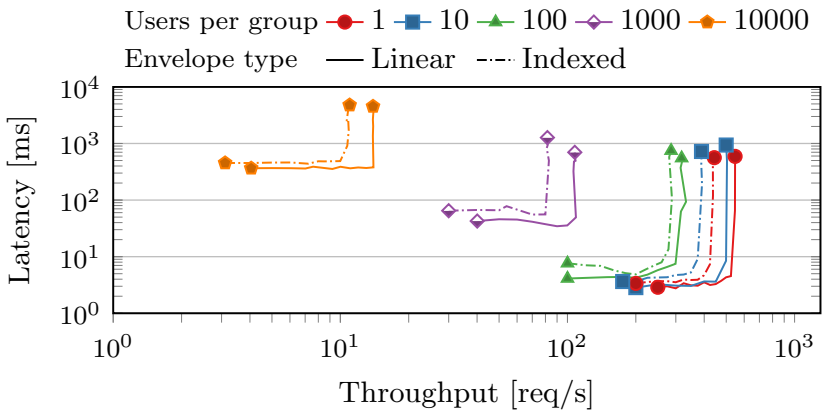


Figure 7.6 – ANO-SKY throughput *vs.* latency plot of enveloping a message for groups of various sizes.

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. 7.7. 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.

7.1.3 REV-SKY

REV-SKY is benchmarked on a cluster of 11 machines with 4-core Intel(R) Core(TM) i7-7567U @ 3.50GHz CPU, and 32 GB of RAM. The machines run Ubuntu 16.04.3 LTS. We consider the median of 10 successive executions for each operation. We do not present standard deviations as these happen to be negligible in all considered experiments. We use a single file of 522 MB, representing the median file size of the satellite images dataset – used later for a macro-benchmark scenario (Figure 7.11a). We vary the block size from 256 KB to 4 MB. We vary the number of super blocks from 1 to 3. As a client comparison baseline, we use a plain encryption scheme, in which similarly to REV-SKY files are split into blocks, but where each block is encrypted using AES.

Figure 7.8a and 7.8b show the REV-SKY execution time for writing and reading data.

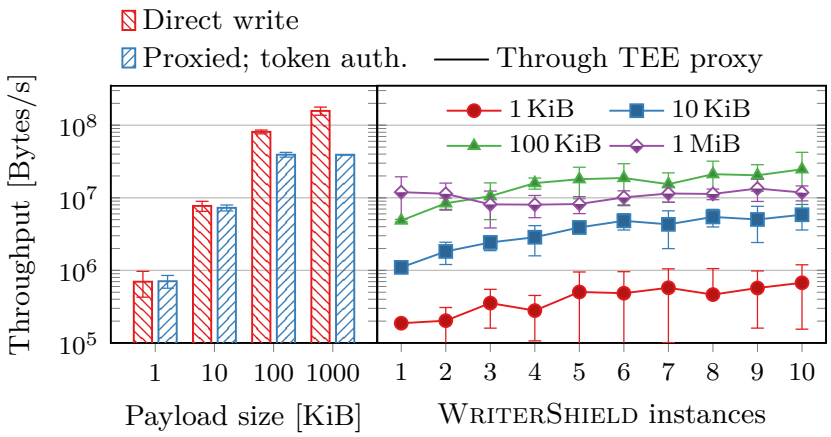


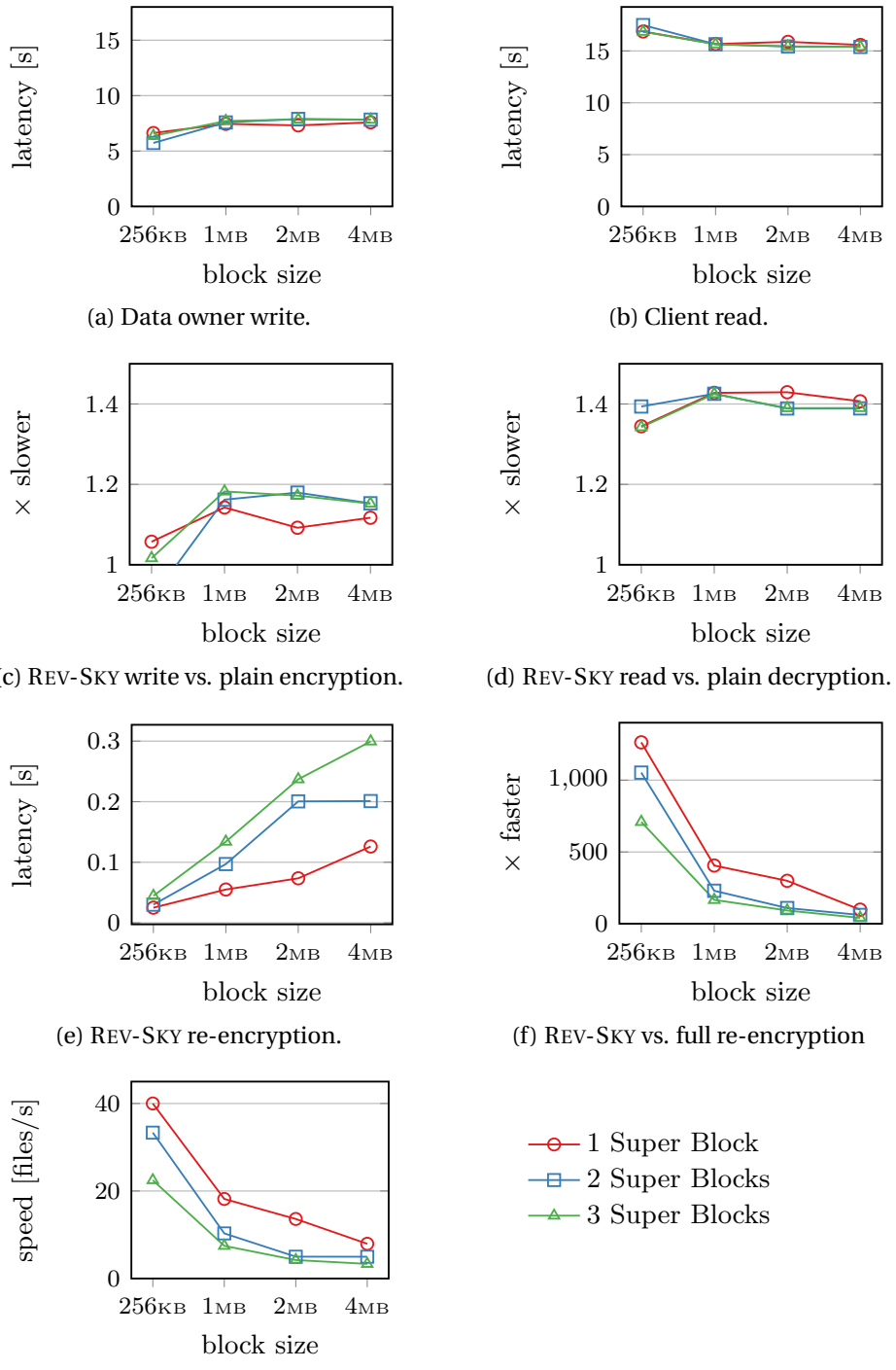
Figure 7.7 – Ano-Sky throughput of writing data to the cloud storage in different ways: directly (baseline), through a TCP proxy using a temporary token for authentication, and through varying number of in-enclave WriterShield instances.

Both operations show nearly constant times for block sizes of 1 MB and larger. Figure 7.8c illustrates that writing a file using Rev-Sky induces a small overhead compared to a simple encryption of each block. Rev-Sky performs 1.11 times slower for writes when considering a single super block of 256 KB. Similarly, Figure 7.8d shows the performance degradation of Rev-Sky when reading a file. Compared to a simple decryption of each block, Rev-Sky introduces a slowdown of 1.38 when considering a single super block of 256 KB. The performance penalty of Rev-Sky at the client side comes mostly from the use of hashing and xor operations performed for the All or Nothing Transformations. There is however only a negligible impact on memory and bandwidth usage for retrieving the file from the cloud storage, as only a small metadata element is fetched in addition to the file's blocks.

As a service provider comparison baseline, we compare Rev-Sky against full re-encryption, in which all blocks of the file are re-encrypted – directly at the provider side within SGX. One should note the chosen baseline already improves over the possible *worst-case* which is downloading all the blocks at owner side, re-encrypting them and pushing the new ciphertexts to the cloud. Figure 7.8e presents the latency of the re-encryption operation. One can notice that for each block size, the time increases nearly linearly with the number of super blocks (with increments on the 10^{-2} scale).

Figure 7.8f shows the performance improvement of Rev-Sky compared to a full re-encryption. Our scheme is 100 times faster when considering a single super block of 4 MB and up to 1,263 times faster with blocks fo 256 KB.

An important property of Rev-Sky is that the revocation operation is independent of the size of the file. Figure 7.8g shows the number of files that can be re-keyed per sec-



(a) Data owner write.

(b) Client read.

(c) REV-SKY write vs. plain encryption.

(d) REV-SKY read vs. plain decryption.

(e) REV-SKY re-encryption.

(f) REV-SKY vs. full re-encryption

(g) REV-SKY revocation throughput

Figure 7.8 – REV-SKY micro-benchmarks at client and storage provider side.

ond using REV-SKY. The throughput ranges from 3.3 files per seconds when using 3 super blocks of 4 MB to 40 files per second when using a single super block of 256 KB.

7.2 Macro-benchmarks

We evaluate now the performance of our three unique contributions in real life scenarios. First, we use the access control traces of the Linux Kernel git repository to simulate collaborative data exchanges over an untrusted hosting infrastructure. Then, to simulate real life usage conditions for anonymous file sharing we leverage the well known Yahoo Cloud Servicing Benchmarks (YCSB). Finally we test the practicality of revocation on large satellite imagery and genomic data.

7.2.1 Access Control using Linux Kernel Git

To capture the performance of the CON-SKY's 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 [Schmidt et al., 2017]. 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 de-enveloping time.

Figure 7.9 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. Us-

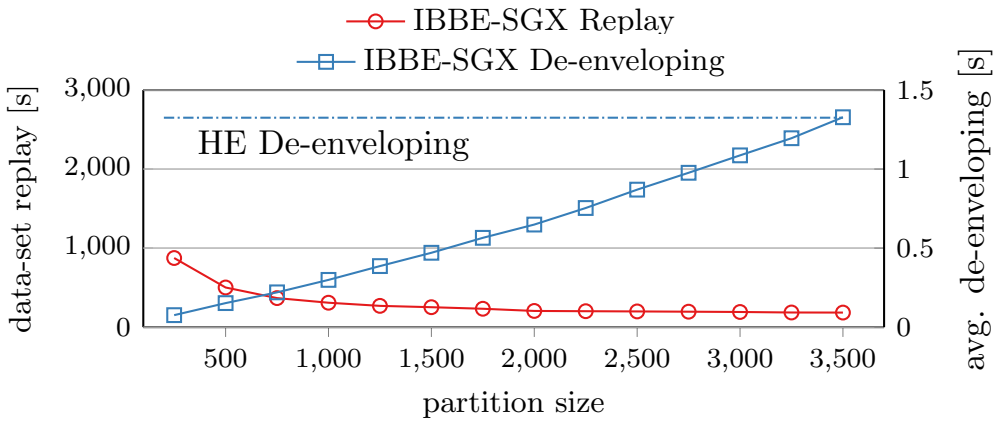


Figure 7.9 – CON-SKY : Measuring total administrator replay time and average user de-enveloping time per different partition sizes using the Linux Kernel ACL data-set.

ing a small partition size of 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, de-envelope 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 administrator performance and user de-envelope time.

7.2.2 Anonymous File Sharing with YCSB

We use YCSB [Cooper et al., 2010] to observe the behavior of ANO-SKY under different usage conditions that are specific to data serving systems. We implemented an interface layer to link the benchmarking tool to ANO-SKY. As our system is not capable of direct-access writes, *update* operations are replaced by read-modify-write (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.

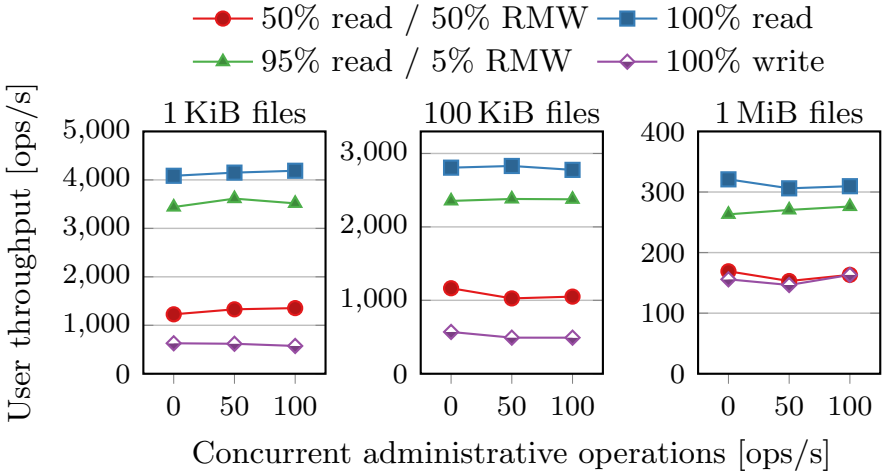
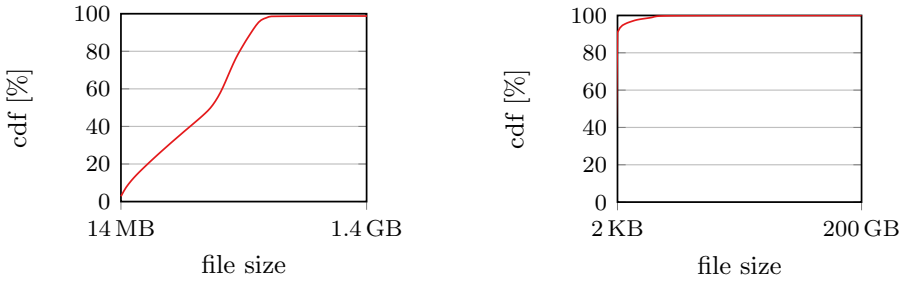


Figure 7.10 – ANO-SKY : User throughput observed by our YCSB-based macro-benchmark, with various file access patterns, varying file sizes, and addition of simultaneous administrative operations.



(a) ESA Satellite Images (b) 1,000 Genome Project

Figure 7.11 – REV-SKY : File sizes distribution

Fig. 7.10 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 KB, 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. 7.7. In a nutshell, we retain that the end-user experience offered by ANO-SKY is not influenced by concurrent administrative operations, and that the overhead of the additional operations required for writing become smaller for larger files.

7.2.3 Revocation of Satellite and Genomic Data

REV-SKY macro-benchmark uses 11 physical machines : one for the admin operations, 4 for the Cassandra cluster, one for the ZooKeeper master node and the remaining 5 for the re-encryption workers. The machines profile is identical to the ones used in REV-SKY micro-benchmark (Section 7.1.3).

We utilize data-sets specific to a satellite imagery provider and a genome processing organization.

For the satellite imagery, we use images acquired by the European Space Agency's satellite Sentinel 2 [GoogleCloud, 2019]. The dataset contains more than four million images, ranging in size from 2 MB to 1.4GB, with an average size of 454 MB and a median size of 522 MB (see Figure 7.11a). The satellite images dataset totals more than 2 PB of data.

For the genome processing organization, we use the data from the *1,000 genomes project* [IntenrationalGenome, 2019]. The dataset contains almost 500,000 files ranging

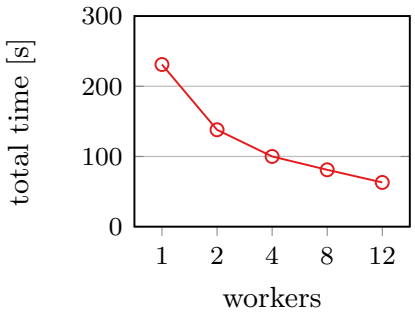


Figure 7.12 – REV-SKY re-encryption scale-out

in size from a few KB to up to 600 GB, with an average of 1.3 GB and a median of 19 MB (see Figure 7.11b). This accounts for a total of more than 650 TB of data.

We first benchmark the scale-out capabilities of REV-SKY when increasing the number of worker nodes. We re-encrypt a sample of 4.3 TB of the satellite data set. This corresponds to a total of 10,000 files. We use a block size of 256 KB, and a single super block per file. As shown in Figure 7.12, the total time for the re-encryption of the data set decreases from 231 seconds for one single worker to 63 seconds when using 12 workers.

We now report the total time for an active revocation when varying the number of files from 100 to 100,000. For both use cases, we generate sample (random) files that mimic the file size distribution of the corresponding datasets, as described in Figure 7.11. Note that we do not need to use the actual data and impose costs for downloading it to its providers, as cryptographic operations have a cost that is independent of the actual content and only the file size matters in our case. For REV-SKY, we use blocks of 256 KB with one single super block and measure the total time for completing the revocation procedure. For full re-encryption, we use blocks of 256 KB and compute the total time by multiplying the time for re-encrypting one single block by the total number of blocks divided by the number of workers. This corresponds to an estimation since we do not take into account the overhead of I/O and coordination service, but it is sufficient to estimate the orders of magnitude. Fig-

		ESA Sentinel 2		Genomic Data	
Files Count	REV-SKY	Size	full re-encryption	Size	full re-encryption
1,000	7.5 s	450 GB	42.2 m	1.3 TB	2.2 h
10,000	59 s	4.3 TB	6.7 h	13 TB	19.2 h
100,000	11.1 m	43.3 TB	2.8 d	132 TB	8.6 d

Figure 7.13 – Results of running REV-SKY on real workloads, using a single 256 KB super block with 12 workers

ure 7.13 shows that REV-SKY enables active revocation of 1,000 files in less than one minute whereas it would take hours using full re-encryption. When considering large volume of data, REV-SKY makes active revocation practicable in few minutes while it would take several days otherwise.

7.3 Closing Remarks

The chapter presented the performance capabilities of our three systems CON-SKY, ANO-SKY, and REV-SKY.

A supplementary benchmark option can be envisioned by integrating the three – currently disjointly evaluated – contributions into a single system. However, the complexity of constructing such *System of systems* goes beyond the functioning of the benchmarking mechanism and requires revisiting the architectural and design concepts of each contribution for a thoughtful integration. We leave such task to future work.

CHAPTER

8 Conclusion

This final chapter concludes the thesis. We recapitulate the context and problem statement, our unique constructions and future work directions. We present two additional research projects that incorporate parts of the presented contributions.

8.1 Contributions Overview

Nevertheless the benefits, public cloud storage miss to give complete security guarantees to end users for their stored data. To address the issue, users can chose to encrypt the data before sending it to the cloud provider. However, encryption complicates the *access control* to the data, reshaping it into a *cryptographic access control* mechanism – usually addressed by the management of the encryption key. State-of-the-art cryptographic access control is not generally well suited for cloud specific workloads (Section 2.4) characterized by large user bases, large data volumetry and as highly dynamic [Garrison et al., 2016]. This thesis presented three unique contributions that use applied cryptographic constructs to tackle efficient *cryptographic access control* for cloud specific workloads :

— CON-SKY enables efficient *confidential* access control by leveraging Trusted Execution Environments (TEE). CON-SKY relies on Identity Based Broadcast Encryption (IBBE) for key management. IBBE is known to achieve small constant size metadata. By leveraging TEE, we change the assumptions of traditional IBBE and derive a new scheme (IBBE-SGX) with faster computational time while maintaining the small constant size metadata benefit. To further optimize user time, CON-SKY introduces a partitioning mechanism for efficient key update. CON-SKY is 1.6 orders of magnitude (OoM) faster than state of the art while producing metadata that is 3 OoM less (Section 7.1.1).

— ANO-SKY enables not only *confidential*, but also *anonymous* access control, made efficient by the use of TEE. ANO-SKY builds upon Anonymous Broadcast Encryption (ANOBE) and similarly to CON-SKY uses TEE to perform a change in assumptions, offering faster computational time and lower storage footprint. In addition, ANO-SKY design accommodates multiple administrative TEE instances, enabling scalability within a large organization deployment. ANO-SKY is 3 OoM better than state-of-the-art ANOBE while an end-to-end system can handle groups of 10,000 users with a throughput of 100,000 key derivations per second per service instance (Section 7.1.2).

— REV-SKY targets access revocation to large data-sets of considerable size through a lightweight re-encryption mechanism. REV-SKY leverages All Or Nothing Transform (AONT) and TEE to re-encrypt only portions of the data directly at the storage provider side. For considerably large file sizes (Section 7.2.3) REV-SKY makes active revocation practicable in few minutes while it would take several days otherwise.

Scientific Dissemination

The presented contributions were published within a number of international research conferences : *DSN* [Contiu et al., 2018a], *SRDS* [Contiu et al., 2019b], and *DAIS* [Contiu et al., 2017]. One scientific collaboration (see Section 8.3) appeared in *Middleware* [Da Silva et al., 2019]. Worth mentioning publications in French National conferences : *Compas* [Contiu et al., 2019a, 2018b], and *RESSI* [Contiu, 2019].

Industrial Impact

Scille is the laureate (July 2019) of a financing from the Region Île-de-France through a PIA [1] project, partially secured for the integration of the original contributions of this thesis within the PARSEC product [Innov'up, 2019].

8.2 Avenues for Future Work

We introduce next three general lines for extensions to this work. We kindly remind the reader that additional in-depth future work items are also given within the Closing Remarks sub-chapters of each of the three contributions (Section 4.4 for CON-SKY, Section 5.7 for ANO-SKY, and Section 6.5 for REV-SKY).

Asymmetric TEE integration

We propose a new research axis for improving the performance of established cryptographic schemes by an *asymmetric* integration of TEE. Cryptographic schemes often have

1. from the french *programme d'investissements d'avenir* : investments for the future.

uneven (*asymmetric*) roles for the interacting actors (e.g., *signer, verifier, prover*). We propose therefore the adaptation of cryptographic schemes with TEE only for the minimal set of roles. Contrary, a *symmetric* integration of TEE means that all the actors in the scheme posses TEE. The same idea of *asymmetric* TEE integration is underlying the change of assumptions of CON-SKY and ANO-SKY, in which only the enveloping operation is outsourced to TEE but not the de-enveloping one. Few examples that could similarly benefit from such an integration are : (1) Attribute Based Encryption (ABE) [Bethencourt et al., 2007] could chose to outsource either the *encryptor* or *decryptor* to TEE; (2) Group Signature Schemes [Ateniese et al., 2000] – either the *signer* or the *verifier*; (3) Zero-Knowledge Proofs [Rackoff and Simon, 1991] – either the *challenger* or the *prover*. Off course, such assumption changes need to be justified by real usage scenarios.

Blockchain

Blockchain like technologies have shown adoption for strengthening the dependability and security guarantees of systems running in untrusted environments. In such lines, our group sharing system could benefit from blockchain in many ways. First, we can employ decentralized ledgers to guarantee traceability for all access control operations, certifying that a user had certain capabilities at a given time [2]. Second, we can decentralize trust over many administrative agents who need to decide together whether to take an administrative action (e.g. add or remove members). Third, blockchain tokens could be utilized as an *incentive* for end-users to provide local storage to a confidential distributed storage system, eliminating the necessity of a cloud provider.

Filesystem Integration

A third axis of research is the integration of the presented cryptographic constructs in a full fledged file system backed by cloud storage. Such systems – notably illustrated by SCFS [Bessani et al., 2014] – require addressing many system challenges and not only key management. First, one needs to bridge the gap between *eventual* consistency usually supported by cloud providers and *strong* consistency required by file systems. Second, an efficient synchronization mechanism needs to keep up-to-date the data shared among end-users, probably through a confidential *publish-subscriber* mechanism [Pires et al., 2016]. Third, in certain usage conditions the data volume can easily become a bottleneck. As such, one might consider an incremental *delta*-changes mechanism enabled with confidentiality. Otherwise, secure *de-duplication* [Li et al., 2016a], garbage collection [Bessani et al., 2014], or *compression* enabled encryption [Zheng et al., 2017] could be envisioned to lower the storage cost.

2. If *anonymity* is desired, traceability can be antagonic to privacy - unless if traceability is performed in a *zero-knowledge* manner, probably using TEE capabilities.

8.3 Scientific Collaborations

The author is part of two additional scientific collaborations that incorporate certain aspects introduced in this thesis :

1. *Privacy-Preserving Edge-Assisted Video Streaming using TEE* [Da Silva et al., 2019]. Quality of experience (QoE) of video streaming can benefit from aggregating provisioned content at edge peers. However, relying on edge peers requires protecting both the identity and viewing preferences among end users. Besides obfuscating video requests, the solution makes use of Trusted Execution Environments (TEE). TEE enabled clients are attested by the mechanism discussed in Sections 4.1.1 and 5.2 and provisioned with long term secrets. Furthermore, renewing the provisioned secrets can be efficiently performed through a cryptographic access control mechanism, CON-SKY or ANO-SKY depending on desired security guarantees.

2. *Improving Tor dependability using TEE.* Tor enables anonymous communication by routing a multi-layer package (called *onion*) through many nodes. Each node on the route only understands one layer of the *onion*. Whenever a route fails, the process restarts with a new *onion*. The proposed solution leverages TEE to re-build *onions* on-the-fly and resume the travel with a newly crafted terminating route. The solution makes use of our TEE attestation and access control mechanisms.

Bibliography

Abu-Libdeh, H., Princehouse, L., and Weatherspoon, H. (2010). Racs: a case for cloud storage diversity. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 229–240. ACM. Not cited.

Ahmad, A., Kim, K., Sarfaraz, M. I., and Lee, B. (2018). Obliviate: A data oblivious filesystem for intel sgx. In *NDSS*. Cited page 72.

Angel, S. and Setty, S. (2016). Unobservable communication over fully untrusted infrastructure. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 551–569. Cited page 26.

Ateniese, G., Camenisch, J., Joye, M., and Tsudik, G. (2000). A practical and provably secure coalition-resistant group signature scheme. In *Annual International Cryptology Conference*, pages 255–270. Springer. Cited page 89.

Ateniese, G., Fu, K., Green, M., and Hohenberger, S. (2006). Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):1–30. Cited page 26.

Aublin, P.-L., Kelbert, F., O'Keeffe, D., Muthukumaran, D., Priebe, C., Lind, J., Krahn, R., Fetzer, C., Eyers, D., and Pietzuch, P. (2018). Libseal: revealing service integrity violations using trusted execution. In *Proceedings of the Thirteenth EuroSys Conference*, page 24. ACM. Cited page 28.

Azab, A. M., Ning, P., Shah, J., Chen, Q., Bhutkar, R., Ganesh, G., Ma, J., and Shen, W. (2014). Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 90–102. ACM. Cited page 12.

Bacis, E., De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Rosa, M., and Samarati, P. (2016). Mix&slice: Efficient access revocation in the cloud. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 217–228. ACM. Cited page 17.

Backes, M., Cachin, C., and Oprea, A. (2005). Lazy revocation in cryptographic file systems. In *Third IEEE International Security in Storage Workshop (SISW'05)*, pages 11–pp. IEEE. Cited page 5.

Bajaj, S. and Sion, R. (2013). Trusteddb: A trusted hardware-based database with privacy and data confidentiality. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):752–765. Cited page 4.

Barker, E., Barker, W., Burr, W., Polk, W., and Smid, M. (2007). NIST special publication 800-57. *NIST Special publication*, 800(57):1–142. Cited page 76.

Barth, A., Boneh, D., and Waters, B. (2006). Privacy in encrypted content distribution using private broadcast encryption. In *International Conference on Financial Cryptography and Data Security*, pages 52–64. Springer. Cited pages 5, 7, 21, 48, 49, 53, 58, 76, 77, and 105.

Băsescu, C., Cachin, C., Eyal, I., Haas, R., Sorniotti, A., Vukolić, M., and Zachevsky, I. (2012). Robust data sharing with key-value stores. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–12. IEEE. Not cited.

Behl, J., Distler, T., and Kapitza, R. (2017). Hybrids on steroids: Sgx-based high performance bft. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 222–237. ACM. Cited page 27.

Bellare, M. and Namprempre, C. (2000). 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*, pages 531–545. Springer. Cited page 58.

Bermbach, D., Klems, M., Tai, S., and Menzel, M. (2011). Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 452–459. IEEE. Not cited.

Bessani, A., Correia, M., Quaresma, B., André, F., and Sousa, P. (2013). Depsky: dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4):12. Cited pages 4, 6, and 31.

Bessani, A. N., Mendes, R., Oliveira, T., Neves, N. F., Correia, M., Pasin, M., and Verissimo, P. (2014). Scfs: A shared cloud-backed file system. In *USENIX Annual Technical Conference*, pages 169–180. Cited pages 4, 29, and 89.

Bethencourt, J., Sahai, A., and Waters, B. (2007). Ciphertext-policy attribute-based encryption. In *2007 IEEE symposium on security and privacy (SP'07)*, pages 321–334. IEEE. Cited pages 25, 30, and 89.

Boneh, D., Boyen, X., and Goh, E.-J. (2005a). Hierarchical identity based encryption with constant size ciphertext. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 440–456. Springer. Cited pages 25 and 44.

Boneh, D. and Franklin, M. (2001). Identity-based encryption from the Weil pairing. In *CRYPTO 2001*, pages 213–229. Springer. Cited pages 4 and 18.

Boneh, D., Gentry, C., and Waters, B. (2005b). Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Crypto*, volume 3621, pages 258–275. Springer. Cited pages 5, 19, 29, and 35.

Boneh, D., Lewi, K., Montgomery, H., and Raghunathan, A. (2013). Key homomorphic prfs and their applications. In *Annual Cryptology Conference*, pages 410–428. Springer. Cited page 30.

Boneh, D. and Lipton, R. J. (1996). A revocable backup system. In *USENIX Security Symposium*, pages 91–96. Cited page 26.

Boneh, D., Sahai, A., and Waters, B. (2011). Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273. Springer. Cited page 25.

Boneh, D., Shen, E., and Waters, B. (2006). Strongly unforgeable signatures based on computational Diffie-Hellman. In *International Workshop on Public Key Cryptography*, pages 229–240. Springer. Cited page 21.

Bos, J. W., Halderman, J. A., Heninger, N., Moore, J., Naehrig, M., and Wustrow, E. (2014). Elliptic curve cryptography in practice. In *International Conference on Financial Cryptography and Data Security*, pages 157–175. Springer. Cited page 11.

Bowers, K. D., Juels, A., and Oprea, A. (2009). Hail: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 187–198. ACM. Not cited.

Boyko, V. (1999). On the security properties of oaep as an all-or-nothing transform. In *Annual International Cryptology Conference*, pages 503–518. Springer. Cited page 22.

Brenner, S., Wulf, C., Goltzsche, D., Weichbrodt, N., Lorenz, M., Fetzer, C., Pietzuch, P., and Kapitza, R. (2016). Securekeeper: confidential zookeeper using intel sgx. In *Proceedings of the 17th International Middleware Conference*, page 14. ACM. Cited pages 6 and 27.

Bulck, J. V., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T. F., Yarom, Y., and Strackx, R. (2018). Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 991–1008. Cited page 76.

Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M., and Pinkas, B. (1999). Multicast security: A taxonomy and some efficient constructions. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 2, pages 708–716. IEEE. Cited page 26.

Chown, P. (2002). Advanced encryption standard (aes) ciphersuites for transport layer security (tls). Cited page 10.

Chung, J. Y., Joe-Wong, C., Ha, S., Hong, J. W.-K., and Chiang, M. (2015). Cyrus: Towards client-defined cloud storage. In *Proceedings of the Tenth European Conference on Computer Systems*, page 17. ACM. Cited pages 4 and 31.

CNBC (2018). Amazon investigating claims of employees leaking data for bribes. https://www.cnbc.com/2018/09/17/amazon-investigating-claims-of-employees-leaking-data-for-bribes.html. Cited page 3.

CNN (2019). Netflix adds 9 million paying subscribers, but stock falls. https://edition.cnn.com/2019/01/17/media/netflix-earnings-q4/index.html. Cited page 5.

Computing, N. (2018). Cloud storage adoption soars in the workplace. https://www.networkcomputing.com/data-centers/cloud-storage-adoption-soars-workplace. Cited page 3.

Contiu, S. (2019). parsecl.cloud : Secure file storage and sharing. In *Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information (RESSI)*. Cited page 88.

Contiu, S., Leblond, E., and Réveillère, L. (2017). Benchmarking cryptographic schemes for securing public cloud storages. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 163–176. Springer. Cited pages 12 and 88.

Contiu, S., Pires, R., Vaucher, S., Pasin, M., Felber, P., and Réveillère, L. (2018a). Ibbe-sgx: Cryptographic group access control using trusted execution environments. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 207–218. IEEE. Cited pages 33 and 88.

Contiu, S., Réveillère, L., and Rivière, E. (2018b). Hybrid revocation using trusted ' execution environments. In *Compas*. Cited page 88.

Contiu, S., Réveillère, L., and Rivière, E. (2019a). Multishot all or nothing transform for efficient revocation. In *Compas*. Cited page 88.

Contiu, S., Vaucher, S., Pires, R., Pasin, M., Felber, P., and Réveillère, L. (2019b). Anonymous and confidential file sharing over untrusted clouds. Cited pages 47 and 88.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM. Cited pages 57 and 83.

Costan, V. and Devadas, S. (2016). Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118. Cited pages 6, 12, 13, 50, 62, and 63.

Costan, V., Lebedev, I., and Devadas, S. (2016). Sanctum: Minimal hardware extensions for strong software isolation. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 857–874. Cited page 12.

Da Silva, S., Ben Mokhtar, S., Contiu, S., Négru, D., Réveillère, L., and Rivière, E. (2019). Privatube : Privacy-preserving edge-assisted video streaming. In *Proceedings of the 20th International Middleware Conference*. ACM. Cited pages 88 and 90.

Delerablée, C. (2007). *Identity-Based Broadcast Encryption with Constant Size Ciphertexts and Private Keys*, pages 200–215. Springer. Cited pages 6, 19, 36, 39, and 40.

Delerablée, C., Paillier, P., and Pointcheval, D. (2007). Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. *Pairing-Based Cryptography–Pairing 2007*, pages 39–59. Cited pages 4, 5, 19, 35, and 44.

Dobre, D., Viotti, P., and Vukolić, M. (2014). Hybris: Robust hybrid cloud storage. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14. ACM. Cited page 4.

Dolev, D. and Yao, A. (1983). On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208. Cited page 17.

Dwyer III, S. J., Weaver, A. C., and Hughes, K. K. (2004). Health insurance portability and accountability act. *Security Issues in the Digital Medical Enterprise*, 72(2):9–18. Cited page 47.

El Mrabet, N. and Joye, M. (2017). *Guide to pairing-based cryptography*. Chapman and Hall/CRC. Cited page 36.

Ellison, C. and Schneier, B. (2000). Ten risks of PKI: What you're not being told about public key infrastructure. *Computer*, 16(1):1–7. Cited page 18.

Fazio, N. and Perera, I. M. (2012). Outsider-anonymous broadcast encryption with sublinear ciphertexts. In *International Workshop on Public Key Cryptography*, pages 225–242. Springer. Cited page 20.

Ferguson, N. and Schneier, B. (2003). *Practical cryptography*, volume 23. Wiley New York. Cited pages 9, 10, and 18.

Fiat, A. and Naor, M. (1993). Broadcast encryption. In *Annual International Cryptology Conference*, pages 480–491. Springer. Cited pages 19 and 26.

Fisch, B., Vinayagamurthy, D., Boneh, D., and Gorbunov, S. (2017). Iron: functional encryption using Intel SGX. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 765–782. ACM. Cited pages 26, 27, 44, and 58.

Forbes (2018). Dropbox is doing well, but looks rich in the face of industry headwinds. https://www.forbes.com/sites/greatspeculations/2018/05/21/dropbox-is-doing-well-but-looks-rich-in-the-face-of-industry-headwinds/. Cited page 2.

Garrison, W. C., Shull, A., Myers, S., and Lee, A. J. (2016). On the practicality of cryptographically enforcing dynamic access control policies in the cloud. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 819–838. IEEE. Cited pages 5, 17, 18, 25, and 87.

GeekWire (2018). Amazon tops 600k worldwide employees for the 1st time, a 13% jump from a year ago. https://www.geekwire.com/2018/amazon-tops-600k-worldwide-employees-1st-time-13-jump-year-ago/. Cited page 5.

Genomics, G. (2018). Genomic data is going google. https://cloud.google.com/genomics/resources/google-genomics-whitepaper.pdf. Cited page 5.

Goh, E.-J., Shacham, H., Modadugu, N., and Boneh, D. (2003). Sirius: Securing remote untrusted storage. In *NDSS*, volume 3, pages 131–145. Cited page 4.

GoogleCloud (2019). Sentinel-2 data. https://cloud.google.com/storage/docs/public-datasets/sentinel-2. Cited page 84.

Goyal, V., Pandey, O., Sahai, A., and Waters, B. (2006). Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm. Cited page 25.

Granlund, T. et al. (1991). GMP, the GNU multiple precision arithmetic library. Cited page 43.

Green, M. and Ateniese, G. (2007). Identity-based proxy re-encryption. In *International Conference on Applied Cryptography and Network Security*, pages 288–306. Springer. Cited page 26.

Green, M., Hohenberger, S., Waters, B., et al. (2011). Outsourcing the decryption of abe ciphertexts. In *USENIX Security Symposium*, volume 2011. Cited page 26.

Guardian, T. (2016). Dropbox hack leads to leaking of 68m user passwords on the internet. https://www.theguardian.com/technology/2016/aug/31/dropbox-hack-passwords-68m-data-breach. Cited page 3.

Gueron, S. and Kounavis, M. E. (2010). Intel® carry-less multiplication instruction and its usage for computing the gcm mode. *White Paper*. Cited page 10.

Han, S., Shen, H., Kim, T., Krishnamurthy, A., Anderson, T., and Wetherall, D. (2015). Metasync: File synchronization across multiple untrusted storage services. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, pages 83–95. Cited page 4.

Hu, V. C., Kuhn, D. R., Ferraiolo, D. F., and Voas, J. (2015). Attribute-based access control. *Computer*, 48(2):85–88. Cited page 14.

Innov'up (2019). Innov'up leader pia. http://leaderpia.iledefrance.fr/innov-up-leader-pia. Cited page 88.

Intel (2017a). *Intel Software Guard Extensions SDK Developer Reference for Linux OS*. Cited page 43.

Intel (2017b). Intel software guard extensions SSL. https://github.com/intel/intel-sgx-ssl. Cited pages 43 and 56.

IntenrationalGenome (2019). Igsr: The international genome sample resource. https://www.internationalgenome.org/. Cited page 84.

Junqueira, F. and Reed, B. (2013). *ZooKeeper: Distributed Process Coordination*. O'Reilly Media, Inc., 1st edition. Cited page 66.

Kaliski, B. and Staddon, J. (1998). Rfc2437: Pkcs# 1: Rsa encryption. Cited page 22.

Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., and Fu, K. (2003). Plutus: Scalable secure file sharing on untrusted storage. In *Fast*, volume 3, pages 29–42. Cited pages 5 and 16.

Kim, T., Park, J., Woo, J., Jeon, S., and Huh, J. (2019). Shieldstore: Shielded in-memory key-value storage with sgx. In *Proceedings of the Fourteenth EuroSys Conference 2019*, page 14. ACM. Cited page 28.

Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209. Cited page 11.

Koh, J. S., Bellovin, S. M., and Nieh, J. (2019). Why joanie can encrypt: Easy email encryption with easy key management. In *Proceedings of the Fourteenth EuroSys Conference 2019*, page 2. ACM. Cited page 4.

Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., et al. (2000). Oceanstore: An architecture for global-scale persistent storage. In *ACM SIGARCH Computer Architecture News*, volume 28, pages 190–201. ACM. Cited pages 4 and 30.

Lee, S., Shih, M.-W., Gera, P., Kim, T., Kim, H., and Peinado, M. (2017). Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 557–574. Cited pages 13 and 44.

Li, J., Qin, C., Lee, P. P., and Li, J. (2016a). Rekeying for encrypted deduplication storage. In *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*, pages 618–629. IEEE. Cited pages 4, 6, 30, and 89.

Li, M., Qin, C., Li, J., and Lee, P. P. (2016b). Cdstore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal. *IEEE Internet Computing*, 20(3):45–53. Cited pages 4 and 30.

Libert, B., Paterson, K. G., and Quaglia, E. A. (2012). Anonymous broadcast encryption: Adaptive security and efficient constructions in the standard model. In *International Workshop on Public Key Cryptography*, pages 206–224. Springer. Cited pages 21, 48, 53, and 58.

Lohmann, N. et al. (2018). JSON for modern C++. https://github.com/nlohmann/json. Cited page 56.

Lynn, B. et al. (2006). PBC library. https://crypto.stanford.edu/pbc/. Cited page 43.

MacKenzie, P., Reiter, M. K., and Yang, K. (2004). Alternatives to non-malleability: Definitions, constructions, and applications. In *Theory of Cryptography Conference*, pages 171–190. Springer. Cited page 21.

MarketWatch (2016). Cloud storage market worth 74.94 billion usd by 2021. http://www.marketwatch.com/story/cloud-storage-market-worth-7494-billion-usd-by-2021-2016-09-06-72033123. Cited page 2.

Matetic, S., Ahmed, M., Kostiainen, K., Dhar, A., Sommer, D., Gervais, A., Juels, A., and Capkun, S. (2017). {ROTE}: Rollback protection for trusted execution. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1289–1306. Cited page 71.

Matetic, S., Schneider, M., Miller, A., Juels, A., and Capkun, S. (2018). Delegatee: Brokered delegation using trusted execution environments. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1387–1403. Cited page 28.

Minio, Inc. (2019). Minio: Private cloud storage. https://www.minio.io/. Cited page 57.

Mokhtar, S. B., Boutet, A., Felber, P., Pasin, M., Pires, R., and Schiavoni, V. (2017). X-search: revisiting private web search using intel sgx. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 198–208. ACM. Cited page 28.

MongoDB Inc. (2018). MongoDB C driver. http://mongoc.org/libmongoc/1.12.0/index.html. Cited page 55.

MongoDB Inc. (2019). MongoDB. https://www.mongodb.com/. Cited page 55.

Myers, S. and Shull, A. (2017). Efficient hybrid proxy re-encryption for practical revocation and key rotation. *IACR Cryptology ePrint Archive*, 2017:833. Cited page 6.

Naor, M. and Pinkas, B. (2000). Efficient trace and revoke schemes. In *International Conference on Financial Cryptography*, pages 1–20. Springer. Cited page 26.

Numerama (2018). (original french title) accès aux données : Mounir mahjoubi critique l'extraterritorialité du cloud act américain. https://www.numerama.com/politique/434095-acces-aux-donnees-mounir-mahjoubi-critique-lextraterritorialite-du-cloud-act-americain.html. Cited page 3.

Peterson, Z. N., Burns, R. C., Herring, J., Stubblefield, A., and Rubin, A. D. (2005). Secure deletion for a versioning file system. In *FAST*, volume 5. Cited page 26.

Pires, R., Goltzsche, D., Mokhtar, S. B., Bouchenak, S., Boutet, A., Felber, P., Kapitza, R., Pasin, M., and Schiavoni, V. (2018). CYCLOSA: Decentralizing private web search through SGX-based browser extensions. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 467–477. Cited page 56.

Pires, R., Pasin, M., Felber, P., and Fetzer, C. (2016). Secure content-based routing using intel software guard extensions. In *Proceedings of the 17th International Middleware Conference*, page 10. ACM. Cited pages 4, 27, and 89.

Poddar, R., Boelter, T., and Popa, R. A. (2016). Arx: A strongly encrypted database system. *IACR Cryptology ePrint Archive*, 2016:591. Cited page 4.

Popa, R. A., Lorch, J. R., Molnar, D., Wang, H. J., and Zhuang, L. (2011a). Enabling security in cloud storage slas with cloudproof. In *USENIX Annual Technical Conference*, volume 242, page 14. Cited pages 4 and 29.

Popa, R. A., Redfield, C., Zeldovich, N., and Balakrishnan, H. (2011b). Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM. Cited page 4.

Rackoff, C. and Simon, D. R. (1991). Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Annual International Cryptology Conference*, pages 433–444. Springer. Cited page 89.

Resch, J. K. and Plank, J. S. (2011). Aont-rs: blending security and performance in dispersed storage systems. In *Proceedings of the 9th USENIX conference on File and stroage technologies*, pages 14–14. USENIX Association. Cited page 4.

Reuters (2017). Equifax breach could be most costly in corporate history. https://www.reuters.com/article/us-equifax-cyber/equifax-breach-could-be-most-costly-in-corporate-history-idUSKCN1GE257. Cited page 3.

Rivest, R. L. (1997). All-or-nothing encryption and the package transform. In *International Workshop on Fast Software Encryption*, pages 210–218. Springer. Cited pages 7 and 22.

Rösler, P., Mainka, C., and Schwenk, J. (2018). 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)*, pages 415–429. IEEE. Cited pages 4 and 26.

Ruoti, S., Andersen, J., Hendershot, T., Zappala, D., and Seamons, K. (2016). Private webmail 2.0: Simple and easy-to-use secure email. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 461–472. ACM. Cited page 4.

Russinovich, M. (2017). Introducing azure confidential computing. https://azure.microsoft.com/en-us/blog/introducing-azure-confidential-computing/. Cited page 23.

Sahai, A., Seyalioglu, H., and Waters, B. (2012). Dynamic credentials and ciphertext delegation for attribute-based encryption. In *Annual Cryptology Conference*, pages 199–217. Springer. Cited page 26.

Sandhu, R. S. (1998). Role-based access control. In *Advances in computers*, volume 46, pages 237–286. Elsevier. Cited page 14.

Schmidt, P. et al. (2017). Linux kernel git revision history. https://www.kaggle.com/philschmidt/linux-kernel-git-revision-history. Cited page 82.

Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G., and Russinovich, M. (2015). Vc3: Trustworthy data analytics in the cloud using sgx. In *2015 IEEE Symposium on Security and Privacy*, pages 38–54. IEEE. Cited pages 6 and 27.

Seybert, H. and Reinecke, P. (2014). Internet and cloud services–statistics on the use by individuals. *Eurostat, Statistics in focus*, 16:2014. Cited page 2.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613. Cited pages 29 and 31.

Shipilev, A. et al. (2018). JMH: Java microbenchmark harness. https://openjdk.java.net/projects/code-tools/jmh/. Cited page 57.

Stefanov, E., van Dijk, M., Juels, A., and Oprea, A. (2012). Iris: A scalable cloud file system with efficient integrity checks. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 229–238. ACM. Cited page 32.

Stefanov, E., Van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., and Devadas, S. (2013). Path oram: an extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 299–310. ACM. Cited page 72.

Stinson, D. R. (2005). *Cryptography: theory and practice*. Chapman and Hall/CRC. Cited pages 9, 18, and 26.

Tang, H., Liu, F., Shen, G., Jin, Y., and Guo, C. (2015). Unidrive: Synergize multiple consumer cloud storage services. In *Proceedings of the 16th Annual Middleware Conference*, pages 137–148. ACM. Cited pages 4 and 31.

Toman, T. (2017). Our data from space lives in google cloud. https://www.planet.com/pulse/planets-data-from-space-lives-in-google-cloud/. Cited page 5.

Tripwire (2016). Massive leak of celebrity nude photos calls cloud security into question. https://www.tripwire.com/state-of-security/vulnerability-management/massive-leak-of-celebrity-nude-photos-calls-apples-icloud-security-into-question/. Cited page 3.

Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T. F., Yarom, Y., and Strackx, R. (2018). Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 991–1008. Cited page 13.

Vaucher, S., Pires, R., Felber, P., Pasin, M., Schiavoni, V., and Fetzer, C. (2018). SGX-aware container orchestration for heterogeneous clusters. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 730–741. Cited page 57.

Wallner, D., Harder, E., Agee, R., et al. (1999). Key management for multicast: Issues and architectures. Technical report, RFC 2627. Cited page 26.

Wang, F., Mickens, J., Zeldovich, N., and Vaikuntanathan, V. (2016). Sieve: Cryptographically enforced access control for user data in untrusted clouds. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 611–626. Cited page 30.

Wang, X., Yin, Y. L., and Yu, H. (2005). Finding collisions in the full sha-1. In *Annual international cryptology conference*, pages 17–36. Springer. Cited page 10.

Waters, B. (2005). Efficient identity-based encryption without random oracles. In *Eurocrypt*, volume 3494, pages 114–127. Springer. Cited page 18.

Wilcox-O'Hearn, Z. and Warner, B. (2008). Tahoe: the least-authority filesystem. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 21–26. ACM. Cited pages 4 and 29.

Wu, Z., Butkiewicz, M., Perkins, D., Katz-Bassett, E., and Madhyastha, H. V. (2013). Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 292–308. ACM. Not cited.

Zheng, W., Li, F., Popa, R. A., Stoica, I., and Agarwal, R. (2017). Minicrypt: Reconciling encryption and compression for big data stores. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 191–204. ACM. Cited page 89.

Zimmermann, P. R. (1995). *The official PGP user's guide*. MIT press. Cited pages 18 and 20.

List of Figures

List of Tables