UNIVERSITY OF TRENTO - Italy

Department of Information Engineering and Computer Science

Master's degree in
Computer Science

FINAL DISSERTATION

A PRAGMATIC APPROACH TO HANDLE "HONEST BUT CURIOUS" CLOUD SERVICE PROVIDERS: CRYPTOGRAPHIC ENFORCEMENT OF DYNAMIC ACCESS CONTROL POLICIES

Supervisor

Prof. Silvio Ranise

Student

Stefano Berlato

Co-supervisor

Dr. Roberto Carbone

Accademic year 2018/2019

I would like to express my most profound gratitude to my supervisors, Prof. Silvio Ranise and Dr. Roberto Carbone, for their invaluable advice and mentoring in the development of this thesis. I am sincerely grateful to all those working in the Security&Trust unit in FBK for their warm welcome and precious support within but also outside the office. Thank you for the Iranian lessons and the first-class football matches; I already filled the Doodle, see you on Monday.

I deem my university career to have been memorable not for the personal gratifications I received, but instead for all the amazing friends and people I met. I want to think of my crazy roommates Daniele, Fabio and Nicolò and the many university mates for making me enjoy all these years. A particular mention to "I Più Belli di Povo"; each of you contributed in a unique way to my journey and is still an unending source of inspiration. Thank you Lorenzo for making me love your Territorio e Piatti Tipici with unceasing sponsoring, Giacomo for making me believe in the Force, Andrei for catching my rockets in Teeworlds, Mahed for your witty appreciations in our lovely bickerings, Andrea for opening my eyes to the real value of Totti, Paolo for our thrilling games at Universal Paperclips during the equally fantastic lessons of P&IPR.

My biggest thank goes to my parents and my family. Thank you, Lisa and Franco, for backing me up both emotionally and concretely. Without you, I would not even be writing this thesis. Thanks also to my relatives that cared about and kept asking about my university studies: Chiara, Martina, Daniele, Stefano, Marzia and all the others, too many to be singularly mentioned.

One of the best things was to go back home on the weekends and meet with the friends I grew up with. Thanks, Giulio, Lele, Joshi and Jacopo for our evenings, football matches, dinners, board games and the many other things that helped me forget, if only for few hours, about projects and deadlines.

Finally, a special thanks to Lorenza, to whom I will be forever indebted for enduring me all these years and not giving me a punch in the head in the (very rare) moments I was unbearable. Thank you for your smiles, jokes, support, love and all the rest; this thesis is also thanks to you. I just hope I can show you how much I appreciated your presence by standing by you, for as long as it will be.

Desidero esprimere la mia più profonda gratitudine ai miei supervisori, il Prof. Silvio Ranise e il Dott. Roberto Carbone, per i loro inestimabili consigli e supervisione nello sviluppo di questa tesi. Sono sinceramente grato a tutti coloro che lavorano nell'unità di Security&Trust in FBK per la loro calorosa accoglienza e il prezioso supporto all'interno, ma anche al di fuori dell'ufficio. Grazie per le lezioni di Iraniano e le eccezionali partite di calcetto; Ho già riempito il Doodle, ci vediamo Lunedì.

Ritengo che la mia carriera universitaria sia stata memorabile non per le gratificazioni ottenute, ma per tutti gli amici e le persone straordinarie che ho incontrato. Voglio ricordare i miei pazzi coinquilini Daniele, Fabio e Nicolò e i tanti compagni universitari per avermi fatto divertire in tutti questi anni. Una menzione particolare a "I Più Belli di Povo"; ognuno di voi ha contribuito in modo unico al mio viaggio ed è ancora una fonte inesauribile di ispirazione. Grazie Lorenzo per avermi fatto amare il tuo Territorio e Piatti Tipici con la tua incessante sponsorizzazione, Giacomo per avermi fatto credere nella Forza, Andrei per prendere al volo i miei razzi a Teeworlds, Mahed per i tuoi arguti apprezzamenti nei nostri adorabili battibecchi, Andrea per avermi aperto gli occhi al vero valore di Totti, Paolo per le nostre emozionanti partite a Universal Paperclips durante le altrettanto fantastiche lezioni di P&IPR.

Il mio più grande ringraziamento va ai miei genitori e alla mia famiglia. Grazie, Lisa e Franco, per avermi sostenuto sia emotivamente che concretamente. Senza di voi non starei nemmeno scrivendo questa tesi. Grazie anche ai parenti che si sono interessati dei miei studi universitari: Chiara, Martina, Daniele, Stefano, Marzia e tutti gli altri, troppi per essere citati singolarmente.

Una delle cose migliori era tornare a casa nei fine settimana e incontrarmi con gli amici con cui sono cresciuto. Grazie Giulio, Lele, Joshi e Jacopo per le nostre serate, partite di calcio, cene, giochi da tavolo e tante altre cose che mi hanno aiutato a dimenticare, anche solo per qualche ora, progetti e scadenze.

Infine, un ringraziamento speciale a Lorenza, a cui sarò per sempre debitore per avermi perdonato tutti questi anni e non avermi dato un pugno in testa nei (rarissimi) momenti in cui ero insopportabile. Grazie per i tuoi sorrisi, scherzi, sostegno, amore e tutto il resto; questa tesi è anche merito tuo. Spero solo di poterti mostrare quanto ho apprezzato la tua presenza stando accanto a te, per tutto il tempo che sarà.

Contents

List of Figures

List of Tables

Glossary

Client any agent that connects to a Cloud Service Provider, either directly or through a Proxy server.

Policy related to Access Control, it defines the set of rules based on which decide whether to grant or not a certain action over a specific resource to a given entity.

Metadata all data related to the policy with the exception of the protected resources. In the case of Role-Based Access Control, metadata are:

- Lists of Users (U), Roles (R) and Permissions (P)
- Set of assignments between users and roles ($UA \subseteq UXR$) and between roles and permissions ($PA \subseteq RXP$)

Dynamic when referring to an Access Control policy, it means that the Lists and the Sets can be effectively changed at run time.

Cryptographic Access Control Scheme the sequence of operations that comprise the protocol for cryptographically enforcing Access Control policies.

Entities the set of active elements involved in the functioning of the in the Cryptographic Access Control Scheme.

Company the organization that is employing a "Honest but Curious" Cloud Service Provider for outsourcing the storage of its files.

Use Case the ground notion for which the Cryptographic Access Control Scheme is needed in this thesis. This consists of the will of a company to store files in the Cloud maintaining the possibility to share them among employees while preserving their confidentiality with respect to the Cloud.

Properties general term to indicate a set of high-level assumptions, requirements and limitations on a specific cryptographic AC scheme.

Concrete Properties the set of low-level assumptions and requirements formulated by us.

Architecture the way entities communicate with each other in order to implement the Cryptographic Access Control Scheme, along with their relationships and physical location of metadata

Concrete Implementation the implementation of the Cryptographic Access Control Scheme that we developed

Abstract

Cryptographic Access Control is the natural solution to allow companies to store sensitive data in the Cloud while preserving confidentiality and the possibility to let employees share resources. While many researches focused on this topic, the vast majority proposed only abstract schemes. Few studies developed even a working prototype and none deeply studied a deployment in a realistic use case by considering concrete issues like keys management, robustness or scalability. In this thesis, we filled the gap between these abstract schemes and an actual deployment. We addressed real-world issues like keys management, robustness and scalability rather than focusing high-level issues only. Concretely, we chose a state-of-the-art cryptographic access control scheme and extended it to deliver a fully-working implementation. We formulated four high-level properties to derive low-level assumptions and requirements. Furthermore, we formalized two degrees of freedom and presented twelve different architectures, explaining in detail advantages and drawbacks. To finalize to our discussion, we implemented a cryptographic access control scheme running in a Docker container and interacting with Amazon Web Services. To cover as many use cases as possible, we designed it so to interface with multiple Cloud Service Providers and support multiple architectures with a simple, if not completely null, configuration effort. Summarizing, our work provided a detailed view of the challenges and conditions that can serve as guidelines also for other future implementations of cryptographic access control schemes.

Keywords [Cryptographic Access Control; Cloud; Implementation]

Chapter 1

Introduction

Innovation can be defined as the implementation of new or significantly revised technologies and architectures aimed at improving products and processes within a company. Since the creation of the very first computers where the most important driver for innovation was the availability of permanent and fast memory, many technological enhancements have been proposed. Almost sixty years ago, the first time-sharing system was developed and followed by the new multiprogramming paradigm. To avoid wastes, many users would simultaneously exploit the expensive computing resources of a single centralized machine without letting it idle. This was a major shift in production and the first notion similar to what is now known as "Cloud Computing". In the eighties, this paradigm was threatened by the commercialization of personal computers. Users would gain control over their operations, customize their working environment and autonomously install software [1]. Gradually, the idea of a centralized mainframe started wavering and productivity flowed in a distributed fashion. However, this introduced other issues regarding the sharing of resources and collaboration among users. Together with the advent of the Internet, this led to the creation of the client-server model and the replacement of mainframes with data centres. In recent years, the managing of these data centres has gotten more difficult and more expensive, also due to the increase in the scale of tasks like storage of files and data analysis. Some of the most crucial challenges to face lie on scalability, maintenance, robustness and overall cost reduction. Because of this situation, there was the need of an innovation introducing a new computational paradigm allowing companies to have an alternative solution to traditional self-managed data centres, but still empowering users to share resources and collaborate together. This is the context in which Cloud Computing ("Cloud" from now on) was first mentioned and conceived.

1.1 Moving Toward the Cloud

The first public use of this term was in 2006 when Google CEO Eric Schmidt stated "I don't think people have really understood how big this opportunity [the Cloud] really is." [2]. Since then, there has been an exponential growth both in quality and quantity of Cloud services. In fact, this innovation stations now at the Slope of Enlightenment stage in the Hype Cycle by Gartner, in the phase of its history where "More instances of how the technology can benefit the enterprise start to crystallize and become more widely understood" [3]. This means that companies are really starting to understand the potentiality of Cloud. Amazon, Microsoft, Google, IBM and many others can be referred to Cloud Service Providers (CSPs) since they offer Cloud services configurable at different levels to cope with the different needs companies have. Today, the Cloud can be defined as a paradigm for

making convenient computing resources ubiquitously available on-demand through the network [4]. The services that CSPs offer range from simple hardware infrastructures to complete applications accessible through web browsers or similar interfaces. Thanks to this variety of choices answering the challenges previously described, many companies have moved part or all of their IT related components to the Cloud. A research by the European Union reveals that 56% of European large companies (250 employees or more) and 23% of small ones (10 to 49 employees) are now exploiting CSPs [5]. Also, the public administration of Italy is designing a digitalization plan that includes the use of the Cloud [6]. Mainly, this was chosen to reduce and simplify issues deriving from the management of data centres. We can conclude that both private and public companies have been and are still dislocating on-premises programs and data to remote third-parties.

1.2 New Challenges and Issues

As described before, CSPs are mostly used to let users share resources and collaborate with each other. This is emphasized by the fact that, after e-mails, the most widely used feature with 68% companies exploiting it is the storage of files [5]. Some famous storage services are S3 by Amazon Web Services (AWS), Drive by Google or OneDrive by Microsoft. However, the dislocation of data toward CSPs raises new challenges and issues about privacy and security. While before companies were entirely responsible for the physical and cyber protection of data, now this is not the case anymore. In fact, sensitive data are now under a shared responsibility between the company and the CSP. AWS has a detailed shared responsibility model that regulates this aspect [7]. AWS is responsible for the security *of* the Cloud, while the company for the security *in* the Cloud. We can abstract that the CSP is usually responsible for hardware infrastructure and software. As depicted in Figure 1.1 (page 12) from [7], the company has other responsibilities. Figure 1.1 shows that the most prominent ones are the encryption of data and Identity and Access Management (IAM), a part of which is the designing of Access Control (AC) policies.

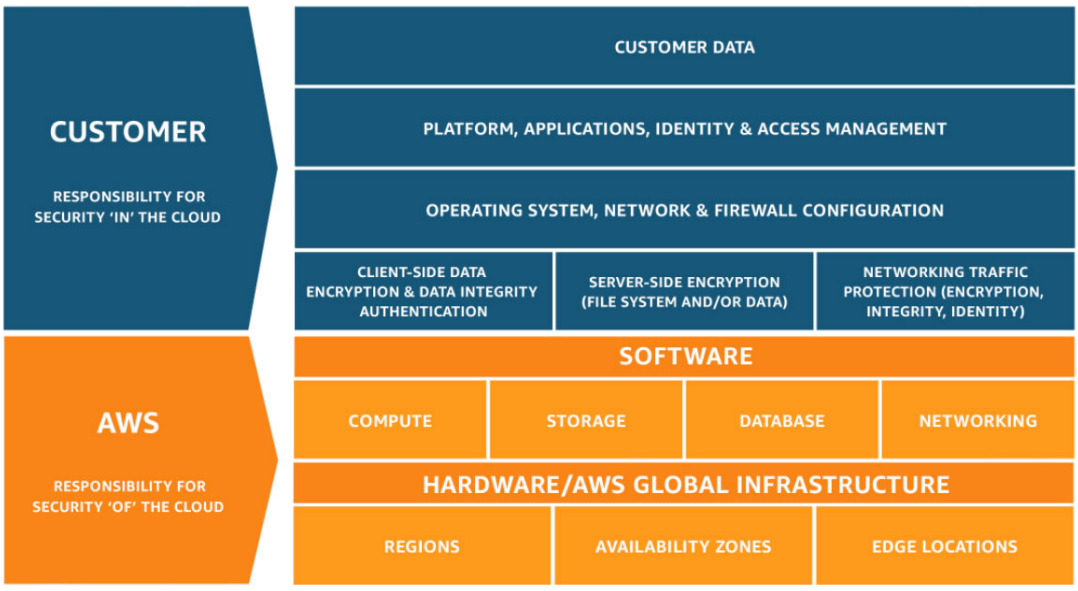


Figure 1.1: AWS Shared Responsibility Model From [7]

1.2.1 Access Control

Regarding the sharing of data among users, the most important task of a company is the configuration of rules to regulate access to its data stored in the CSP. The set of these rules is commonly referred to as an AC policy. Its purpose is to specify whether a given entity can perform a certain action over a specific resource. A suitable example is a scenario in which an employee (entity) from a company wants to read (action) a file containing sensitive data (resource). The pair of action and resource is referred to as permission. For feasibility, policies are usually dynamic, meaning that entities can be added and removed and permissions can be granted and revoked. Depending on which factors are considered, we can formalize different kinds of policies. Attribute-Based AC (ABAC) considers attributes related to the entity, the resource or even the environment, like current time. Identity-Based AC (IBAC) is mainly based on the identity of the entity itself. Role-Based Access Control with no Hierarchy ($RBAC_0$) is one of the simplest and most widely adopted kinds of policy. In the scenario of a company, each employee is assigned to one or more roles that reflect internal qualifications. Then, each role is granted one or more permissions. Employees assume a specific role to carry on their operations as permissions are transitively conveyed.

Many CSPs offer the possibility to configure and then automatically enforce several kinds of AC policies. For instance, AWS and Azure offer an IAM service through which it is possible to define permissions over data and also programs [8, 9]. However, even with the possibility of configuring AC policies, there are still some security concerns that need to be tackled. The first one is misconfiguration. In fact, there have been dozen of famous cases showing badly designed AC policies resulting in data leaks. For instance, in 2017 a database containing personal information of 198 million US voters was found to be publicly accessible on AWS, making up for almost one terabyte of sensitive data [10]. The same year a similar case occurred with the exposure of data related to between 6 and 14 million customers of Verizon by a third-party [11]. An even more resounding event concerns the Pentagon. Although the content was not sensitive, the Pentagon accidentally left a database containing 1.8 billion documents in the Cloud freely accessible. Beside the fact of companies not having understood how to configure AC policies in the Cloud, all these leaks have one thing in common: data were not encrypted, and this is the second major security concern. In fact, Breach Level Index reports that only 4% of all breaches were made useless by the use of encryption [12].

1.2.2 Cryptography in the Cloud

One of the solutions to protect data from malicious attackers or during accidental leaks is the use of cryptography. If the content related to a data breach is encrypted, the leak has very little impact on the responsible company. For this reason, many CSPs integrated into their range of products encryption services to secure stored files. AWS provides Cloud-side default encryption for data stored in S3 [13]. Note that, being done by the CSP, the company has, of course, no control over the process. In fact, cryptographic keys are generated and stored by AWS itself. Another example comes from Microsoft that recently announced OneDrive Personal Vault [14]. Beside enforcing strong authentication and other security features, this service offers encryption at rest and in transit. Again, files are encrypted Cloud-side and eventually synchronized across the various devices connected with the same account.

These precautions enhance the confidence of the companies in storing their files in the Cloud. In fact, they increase the sense of security related to the confidentiality of data with respect to third parties. Still, there is another external entity that has the possibility to read such files. While companies

trust CSPs to properly handle and store data on their behalf, this does not exclude the possibility that CSPs access the content of stored files themselves, for instance, to perform data analysis. In the literature, such behaviour is commonly known and often referred to as "Honest but Curious" or "Partially Trusted" [15, 16]. There are situations in which this conduct may not be acceptable by the company employing the CSP, like when sensitive data or public administrations are involved.

1.3 Cryptographic AC Enforcement as a Solution

In order to still exploit Honest but Curious CSPs for data storage while retaining confidentiality, sharing of files and collaboration among users, *Cryptographic AC* is the perfect solution. Cryptographic AC combines the confidentiality given by cryptography with the application of AC policies over data. Files are encrypted by their owners and are publicly accessible. Decrypting keys act like permissions. As an example, consider again the scenario of an employee who wants to read a file containing sensitive data. An administrator manages the AC policy. Granting to the employee `Read` permission over the file concretely means giving to him/her the decrypting key. The employee can download the encrypted file but cannot read its content, unless he/she is given the keys to decrypt it with. The `Write` permission is granted in the same way, except for the presence of another entity called Reference Monitor (RM). While not being able to read the content of the files, the RM guarantees that `Write` operations are compliant with the AC policy.

Several different cryptographic AC schemes have been proposed in the literature. However, the vast majority is not suitable for a realistic use case because they do not support the dynamicity of the AC policy, in particular, the revocation of keys [17, 18]. Clearly, in a company, it is essential to be able to revoke permissions from employees. Some works considered dynamic revocation of keys, but they did not discuss the computational burden that it implies [19]. Indeed, few researches studied the efficiency and performance of dynamic cryptographic AC schemes [15], often concluding that their scheme is likely to produce "prohibitive overheads" for a real deployment. Even fewer discussed concrete issues like the scalability of the scheme or implemented even a prototype [20, 16]. None focused on studying how to fill the gap between an abstract cryptographic AC scheme and a concrete usable implementation.

1.4 Contribution

Our contribution is a pragmatic approach to deploy a cryptographic AC scheme to handle "Honest but Curious" CSPs. This means addressing concrete issues like keys management, robustness and scalability rather than focusing on abstract schemes only. Our research started from the specific abstract formulation presented in [15] for enforcing dynamic cryptographic AC policies in the Cloud. Even though considering issues like efficiency and realistic use cases, [15] focused on proposing a new abstract schema. Therefore, many aspects like key storage and architecture analysis were omitted or given for granted. This thesis extends [15] by deriving concrete requirements and assumptions to deliver a feasible, flexible, portable and of course secure implementation of the proposed dynamic cryptographic AC scheme. The name of the tool we developed is CryptoAC and it is essentially a Java server running within a Docker container. It offers through either Representational State Transfer (REST) Application Programming Interfaces (APIs) or from the browser interface (see Appendix B) the functionalities expressed by the cryptographic AC scheme.

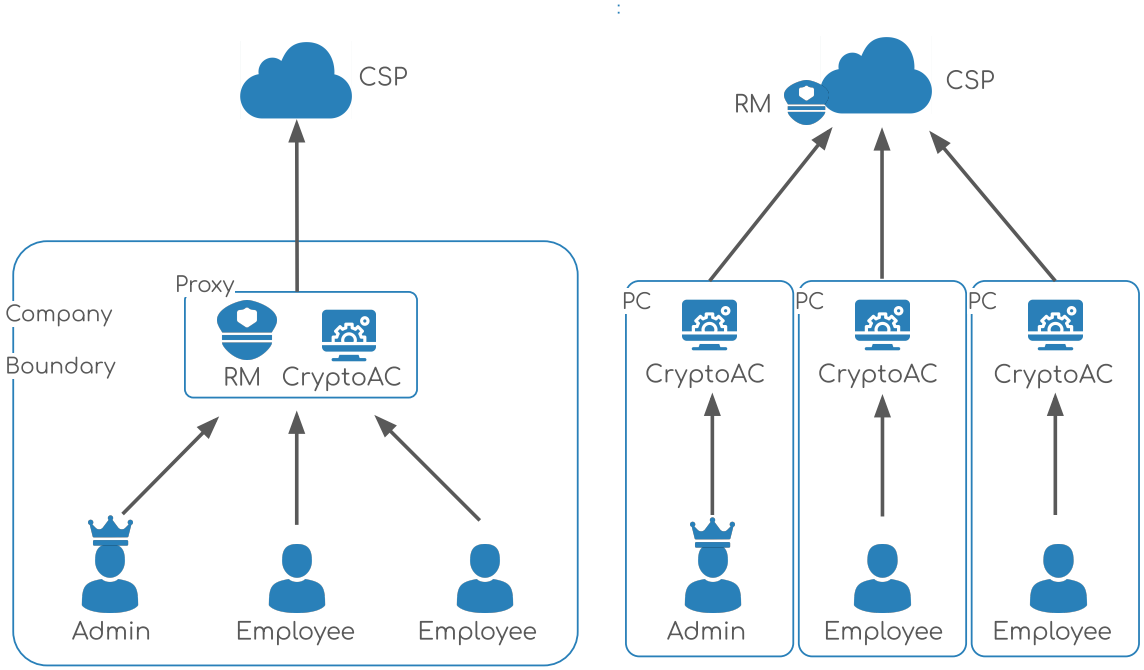
In particular, our contributions are:

- **Preliminary Analysis of the Cryptographic AC Scheme** - Identification of the entities involved in the cryptographic AC scheme presented in [15] (administrator, users, RM, CSP, CryptoAC) for handling Honest but Curious CSPs along with the elicitation of the abstract limitations and assumptions (e.g., the CSP can execute code, secure channels for communications).

- **Concrete Requirements and Assumptions** - Definition of four high-level properties (flexibility, portability, security and privacy) to guarantee a pragmatic approach to implement [15]; formulation of concrete assumptions and requirements (e.g., metadata hiding, multi-CSP support) from both such properties and the previously analyzed abstract assumptions.

- **Architecture** - Identification of two main degrees of freedom regarding how entities connect to the CSP and location of metadata; identification of twelve different architectures in light of these degrees of freedom; further examination of two of these architectures through an analysis of advantages and drawbacks. Figure 1.2 (page 15) reports the two architectures. The first expects a proxy to be deployed by the company within its boundary to mediate all employees' requests, enhancing control. The second, instead, expects direct connections between each employee and the CSP, enhancing scalability.

- **Implementation** - Development of one of the two architectures interacting with a widely used CSP (AWS) based on Docker; Analysis of the compliance of such an implementation with respect to the concrete requirements.



(a) Proxy architecture

(b) Straight architecture

Figure 1.2: Two Architectures

1.5 Thesis Organization

The thesis is organized as follows. In Chapter 2 we present the state of the art in the literature. Even though considering cryptography and AC in general, we focus on discussing related works that propose an actual concrete implementation for the cryptographic AC schemes. In Chapter 3 we present the cryptographic AC scheme we chose as the starting point and explain why we chose it. To do so, we illustrate the basic notions necessary to understand the functioning of cryptographic AC in the Cloud. In particular, we emphasize the dynamic behaviour of the scheme with concrete examples. We also analyze assumptions and limitations either explicitly stated (or inferred) from [15]. In Chapter 4 we define the complete set of assumptions and requirements that ensure a pragmatic approach to properly deploy a cryptographic AC scheme to handle Honest but Curious CSPs. First, we state four high-level properties we want to guarantee and from them we derive concrete requirements. To these, we add other concrete requirements coming from the analysis of the abstract assumptions presented in [15]. As we explain, even though some concepts might have been formulated as assumptions at an abstract level, they become requirements at the implementation level. In Chapter 5 we propose twelve possible architectures for the implementation of the scheme. Initially, we consider how entities and data can be configured and define two degrees of freedom. Then, we analyze how these impact the twelve architectures by discussing the pros and cons. We detail two specific architectures to provide further insights. Finally, in Chapter 6 we present our implementation with AWS and demonstrate how it respects all the requirements and satisfy all the assumptions previously formulated. In Chapter 7, we conclude with some final remarks and in Section 7.1 we present future work.

Chapter 2

Related Work

In this Chapter, we discuss the related work concerning AC and cryptography. First, we introduce the concept of AC and the available kinds of AC policies. then, we restrict the discussion to the Cloud and present some concrete examples in which cryptography is employed. Afterwards, we combine these two concepts by focusing on cryptographic AC. We illustrate different schemes presented in the literature and briefly discuss them in the light of our purpose: a pragmatic approach to concretely deploy a cryptographic AC scheme. As we argue, many practical issues like scalability or robustness were not addressed. This makes such schemes unfeasible for a real deployment. Eventually, we consider works presenting tools developed for implementing cryptographic AC schemes. We show that the majority of these tools was implemented for purposes different than ours (e.g., performance evaluation or proof of concept) and the others were left as (incomplete) prototypes.

2.1 Access Control

The topic of AC in computer science has been widely studied for years and has been employed in many use cases. AC is used in small-scale systems like OSs where permissions are usually assigned through AC Lists. Another example is relational databases (DB) where privileges are assigned to every user and the DB engine administrates permissions. AC is also employed in more complex use cases: companies often have their own AC policy for deciding whether to grant or not a certain permission to a specific employee. In the last years, also CSPs like Amazon and Azure have developed AC policies through which express fine-grained control over the use of their services [8, 9]. Each one of these use cases has peculiar needs that require a customized enforcing of AC. In the literature, several different AC schemes have been formulated and proposed [21, 22, 23]. We do not discuss here the details of these schemes, leaving the references for further insights. Mainly, there are two configurations through which these schemes can be implemented:

- **Full Trust - Centralized**: one possible configuration is to rely on a fully trusted RM. The policy is managed by this entity that decides whether to grant or not a specific permission. In this setup, AC is centralized being the RM logically located in the middle of all communications. Therefore, the RM has full access to the entrusted resources. Unfortunately, this may not be always acceptable and assuming the presence of such an entity is not reasonable in every use case.

- **Partial Trust - Decentralized**: in the second configuration there is not a fully trusted entity. Several studies investigated constraints and limitations of enforcing AC policies in an Honest but

Curious use case, mainly discussing trust issues [24, 25]. Indeed, in such a configuration there may be little or none trust to external or third-parties. Therefore, some researches immediately exclude the presence of a partially trusted RM [26, 27, 28, 29], while others state that some trust, even though minimized, can anyway be assumed to be present [15].

Our use case falls into the second configuration. In fact, it consists of a company outsourcing the storing of its files to an Honest but Curious CSP. First, the company entrusts its files to the CSP that stored them to make them available for next use. Then, the company designs an AC policy to regulate employees' accesses. The CSP is responsible for the enforcing of such a policy. Unfortunately, this use case presents two main problems. First, as already presented in Chapter 1, misconfigurations of AC policies may lead to leaks of data. Second, since the CSP is Honest but Curious, it will read the content of the stored files if given the possibility.

2.2 Crpytography in Remote Storage Solutions

To solve the first problem, many CSPs and other services for remote storage are including automatic encryption features to preserve the confidentiality of stored data. For instance, MongoDB, a popular DB engine, previews in version 4.2 a client-side field level encryption feature [30]. This preserves the confidentiality of data with respect to attackers and the server hosting the DB. In fact, all encryption and decryption operations will occur in the client's driver where keys are also stored. OneDrive by Microsoft, a service for data storage, announced that a new feature (i.e. Personal Vault) will soon be available [14]. Beside enforcing strong authentication and other security measures, it will offer encryption of data at rest and in transit. This means that files will be encrypted by OneDrive before being stored and decrypted when requested by an authorized client. However, Personal Vault will encrypt files Cloud-side, so the content will be readable by OneDrive. AWS already offers automatic encryption feature at rest, but also in this case files are encrypted Cloud-side [13]. AWS offers also the possibility to perform the encryption operation client-side, but keys are either stored in AWS or known only by the client. In the latter case, it is not specified how to distribute the keys to other clients to allow sharing of files.

All these features make stored data more secure by exploiting cryptography. However, they do not combine it with AC. In fact, they either do not consider AC [30] [13, in client-side encryption] or rely again on a central fully-trusted entity (i.e. the CSP) to regulate accesses [14] [13, in Cloud-side encryption]. In the former case, companies have to implement the AC scheme by themselves. In the second case, the content of stored files is readable by the CSP.

2.3 Cryptographic Access Control

This kind of AC policy has been widely studied in the literature for many different use cases, like local filesystems [17] or Cloud-based storage solutions [15]. In this thesis, we focus on cryptographic enforcing of AC policies in the Cloud. Many researchers proposed cryptographic AC schemes, exploiting several algorithms based on identities, predicates, roles or attributes. Goyal et al. developed a scheme for enforcing AC through Attribute-Based Encryption (ABE) [17]. Users can delegate their own permissions, but the scheme does not handle revocation of keys. This makes the whole scheme not feasible for a company where permissions need also to be revoked. Müller et al. proposed a similar scheme while also avoiding the disclosure of the policy itself. Still, the dynamicity of a real use

case was not taken into account [18]. Other works considered revocation of permissions, but many of them did not discuss the computational burden that this implies [19]. Few indeed focused on the practicality of the scheme rather than on the features that it includes. Garrison et al. [15] studied the feasibility of a simple dynamic cryptographic $RBAC_0$ scheme. Their conclusion is that, even when considering a minimally dynamic scenario, the scheme is likely to produce "prohibitive overheads" for a real deployment.

In fact, the use of a cryptographic AC scheme poses some intrinsic limitations. For instance, it is generally more inefficient with respect to centralized AC schemes. This is due to the overhead of encryption and decryption operations. Also, the revocation of permissions tends to be computationally unsustainable, since it often implies the re-encryption of all related files. Also, the kind of permissions that can be cryptographically enforced are limited in many schemes, usually only to `Read` and `Write` actions over files. Besides these intrinsic limitations, there are many other issues that have been rarely considered when developing a cryptographic AC scheme, like key distribution, scalability and robustness. Mainly, this is due to the fact that the vast majority of these works did not consider a concrete implementation of their scheme. Therefore, many cryptographic AC schemes do not have any contact with realistic use cases. However, in the literature, some works implemented tools and prototypes along with the abstract scheme. Unfortunately, as we present in the next Section, a feasible design for a pragmatic deployment is seldom considered.

2.4 Cryptographic Access Control Implementations

There are few works [20, 16, 29, 31, 32, 33, 34] that presented an implementation for the scheme they propose, and even fewer discussed a real and usable deployment. In fact, the major concern is the analysis of the efficiency of the scheme with respect to `Read` and `Write` actions. Often, the costs of modifications to the policy is not addressed. Moreover, these implementations usually have a fixed architecture that cannot be customized to fit a specific use case.

Fugkeaw and Sato [20] developed a scheme to allow multiple owners to give access to their data to multiple users, following a mixed Attribute-Role based AC cryptographic scheme. They implemented an administrative tool named CLOUD-CAT. Their architecture is composed of four modules responsible for users' authentication and authorization, policy management and enforcement and data en/decryption. They discussed scalability and performance in `Read` and `Write` actions, showing a nearly linear behaviour for the latter. However, they did not evaluate the advantages and drawbacks of the architecture, neither they discussed the way these components should be deployed by the company or how they should interact with the CSP. Also, they did not provide alternative designs for a flexible and adaptable deployment. In fact, the set up of many components may carry too much overhead for small companies, making the whole scheme unappealing. The architecture also contains a Single Point Of Failure (SPOF) that makes it less robust. Lastly, the authors did not formulate any requirement or assumption on the use case they operate in or on third parties (e.g., assuming that the CSP is "Honest but Curious").

Zhou et al. [16] supported instead this kind of assumption. They enforced RBAC using asymmetric bilinear groups and Identity-Based Signature (IBS) to certificate data and their origin. In their architecture, a public CSPs stores data while a private one within the company holds metadata. Users communicate only with the public CSPs for accessing data, while the administrator interacts with the private one. This has to be deployed internally, and again this solution may be unappealing for small

companies. In their construction, the public CSPs is supposed to run cryptographic operations on behalf of the users and to communicate with the private one to retrieve the needed metadata. However, the authors did not discuss the feasibility of this communication or the deployability of the architecture. In fact, no analysis of the pros and cons was given supporting the usability of the whole architecture or the effort to consider when configuring the private CSPs. They developed a concrete implementation of their scheme just for conducting an analysis on the operation time for `Read` and `Write` action.

Zarandioon et al. [29] proposed a scheme with a focus on key updating. The authors also emphasized privacy with respect to users' policies and actions, enabling anonymous access to resources stored by the CSPs. In the architecture, the RM is not considered. In fact, users writing on files have to digitally sign them, and the signatures are validated by future readers. The authors implemented a tool interacting with AWS S3, also providing an interface so that further CSPs can be supported. However, practical issues are not taken into account, like a user flooding the CSP with spurious file uploads. Concrete costs like monetary expenses are not considered as well, and no analysis of limitations, assumptions or requirements is performed. The architecture is fixed and cannot be modified to accommodate multiple use cases.

FADE is a proof of concept prototype presented by Tang et al. [31]. Their work focused on the assured deletion of files from the Cloud. The architecture comprehends a quorum of key managers deployed as a centralized trusted entity within the company. However, the authors did not take into account the effort and overhead this may bring on the company. Users interact with a FADE client that can be locally run or as a proxy. This is the first scheme allowing to slightly modify the architecture. However, a central entity has anyway to be present, reducing the overall robustness (SPOF). Multiple CSPs can be supported, and performances and monetary costs were analyzed. However, each file is associated with a single policy, mining the scalability and maintainability of the whole AC scheme. The authors also extended the scheme with a more traditional Attribute-based AC, but no concrete implementation is given for such extension.

Ghita et al. [32] implemented a cryptographic RBAC scheme using ABE. Even though they developed a working prototype, many pragmatic aspects were omitted. For instance, in their implementation is not possible to add roles to the policy. This is a tight limitation on the usability of the scheme. The architecture is fixed and presents a centralized trusted entity acting as a proxy. All keys are stored within this proxy, thus an attacker compromising it would gain knowledge of all of them. Moreover, they did not discuss requirements, assumptions or limitations on their use case.

Crypto-DAC is a scheme proposed by Qi & Zheng [33] to enforce cryptographic dynamic AC policies in the Cloud. This work is very similar to the one presented by Garrison et al. [15]. Its peculiarity is to handling revocation through onion encryption. Each time a permission is revoked, the CSP adds an encryption layer with a new key on each affected file. For reading a file, an authorized user has to decrypt each layer until he/she obtains the original content. The administrator can set a threshold related to the number of encryption layers, after which a de-onioning procedure occurs. The authors implemented their scheme and demonstrated how they obtain only slightly worse performances with respect to [15] (i.e. 7.2%) while being able to immediately block access to files by revoked users. Unfortunately, they did not discuss the monetary costs that their onion mechanism produces. Moreover, metadata are necessarily stored in the Cloud, and there is no discussion about flexibility and deployability of the architecture. Actually, the architecture was never taken into ac-

count explicitly, nor assumptions or requirements are ever stated. Their implementation had the only purpose of measuring the efficiency of the scheme.

In [34], Jang-Jaccard argues that ABE applied to AC in the Cloud only exists in the literature and it is difficult to find a real application. For this reason, the author provided an implementation using non-monotonic Ciphertext-Policy ABE. An administrator is responsible for creating keys from attributes and users exploit a client application for interacting with AWS. Unfortunately, the author did not consider issues related to private keys management or other properties like flexibility and portability. In fact, the library the author used did not guarantee the same functionalities on a different platform. Most importantly, the scheme does not support the dynamicity of the policy as it was left as future work.

To summarize, even though approaching the problem from different points of view, the focus of these works is mainly proposing new techniques for enforcing cryptographic AC in the Cloud. Because of this, little space is left for additional analysis on the interactions and relationships between users, administrators and CSPs. Moreover, they lack a pragmatic analysis of the use cases in which their tool or prototype is to be deployed. This means that they did not discuss requirements, limitations or assumptions of their architectures. In order to develop a feasible and actually usable implementation, we took a step further in the comprehension and understanding of the impact of real-world limitations and CSP capabilities over these schemes.

Chapter 3

Background and Preliminary Analysis of the Cryptographic AC Scheme

We now present the basic notions for understanding how cryptography can be used to enforce dynamic AC policies in the Cloud. Since our contribution is a pragmatic approach to deploy a cryptographic AC scheme, we had to select a specific scheme on top of which build our architecture. We chose the one proposed by Garrison et al. in [15]. The authors leveraged a simple policy (i.e. RBAC_0 through Hybrid Encryption) for achieving the best possible performance. This, together with the dynamicity of the policy, allowed them to build the scheme under a realistic use case, making it suitable for our purpose.

In this Chapter, we first illustrate the concepts of Hybrid Encryption and RBAC_0 policy. Then, we present the cryptographic RBAC_0 scheme and we show how it is possible to read and write on files. We list the entities involved in the functioning of the scheme and discuss the dynamicity of the policy. Then, we argue why we decided to adopt this scheme instead of others presented in Chapter 2. Eventually, we describe limitations and assumptions regarding both the relationships among the different entities and the cryptographic RBAC_0 scheme.

3.1 Basic Concepts

Before presenting how cryptographic RBAC_0 works with Hybrid Encryption, we have first to introduce these two basic concepts. Therefore, we first explain how Hybrid Encryption can be exploited to increase the performance of a general encryption scheme. Then, we illustrate the functioning of a RBAC_0 policy at a high-level.

3.1.1 Hybrid Encryption

Hybrid Encryption is a cryptosystem composed of two different cryptographic algorithms combined together, i.e. asymmetric and symmetric. An asymmetric cryptographic algorithm expects a pair of cryptographic keys: a public ($\mathbf{k^{pub}}$) one accessible by everyone and a private ($\mathbf{k^{pri}}$) one known only by its owner. These two keys are mutually uniquely bound. To securely sends a message M, this is encrypted by the sender with $\mathbf{k^{pub}}$ resulting in $\{M\}_{\mathbf{k^{pub}}}$ and then delivered to the recipient. This uses his/her private key $\mathbf{k^{pri}}$ to obtain the original message, as decrypting $\{M\}_{\mathbf{k^{pub}}}$ with $\mathbf{k^{pri}}$ results in M. This is an asymmetric cryptographic algorithm because the encryption and decryption operations occur with two different keys. A symmetric algorithm, instead, expects only one key ($\mathbf{k^{sym}}$) for both operations. Therefore, the sender and the recipient need to share a common secret.

On the one hand, asymmetric cryptographic algorithms do not require the involved entities to agree on any secret, but they have a bad performance for the bulk encryption of a great amount of data. On the other hand, symmetric cryptographic algorithms are more efficient even though they require the involved entities to exchange the secret key. To demonstrate the functioning of a hybrid cryptosystem, consider the following scenario: Alice (A) wants to securely send a message (M) to Bob (B). B has a pair of public-private keys ($\mathbf{k}_B^{\mathbf{pub}}$, $\mathbf{k}_B^{\mathbf{pri}}$). In a Hybrid Encryption cryptosystem, the secure exchange of the message M consists of these steps:

1. A obtains $\mathbf{k}_B^{\mathbf{pub}}$.
2. A generates a symmetric key $\mathbf{k}^{\mathbf{sym}}$.
3. A encrypts M with $\mathbf{k}^{\mathbf{sym}}$ obtaining $\{M\}_{\mathbf{k}^{\mathbf{sym}}}$.
4. A encrypts $\mathbf{k}^{\mathbf{sym}}$ with $\mathbf{k}_B^{\mathbf{pub}}$ obtaining $\{\mathbf{k}^{\mathbf{sym}}\}_{\mathbf{k}_B^{\mathbf{pub}}}$.
5. A sends both $\{M\}_{\mathbf{k}^{\mathbf{sym}}}$ and $\{\mathbf{k}^{\mathbf{sym}}\}_{\mathbf{k}_B^{\mathbf{pub}}}$ to Bob.
6. B decrypts $\{\mathbf{k}^{\mathbf{sym}}\}_{\mathbf{k}_B^{\mathbf{pub}}}$ with $\mathbf{k}_B^{\mathbf{pri}}$ obtaining $\mathbf{k}^{\mathbf{sym}}$. Then, he decrypts $\{M\}_{\mathbf{k}^{\mathbf{sym}}}$ with $\mathbf{k}^{\mathbf{sym}}$ obtaining M.

Symmetric key encryption is orders of magnitude faster than asymmetric algorithms [15]. Therefore, when dealing with big files, this cryptosystem brings a noticeable increase in performance. In fact, Hybrid Encryption is employed in many well-known protocols (e.g., TLS).

3.1.2 RBAC$_0$

Generally speaking, a policy is a set of rules whose purpose is to specify whether a given entity can perform a certain action over a specific resource. Depending on the specific kind of AC scheme, such a policy can be encoded in many different ways. For what concerns a RBAC$_0$ scheme, the status of the policy can be described as follows:

- U is a set of users.
- R is a set of roles.
- P is a set of permissions.
- $UR \subseteq U \times R$ is a set of assignments between U and R.
- $PA \subseteq R \times P$ is a set of assignments between R and P.

Users are assigned to one or more roles. In the context of a company, a role should reflect an internal qualification (e.g., employee or manager). Permissions are assigned to one or more roles by the administrator of the policy. The administrator is the person responsible for the management of the whole policy. Users assume the identity of a role to access the permissions needed to finalize their operations (e.g., read a file). More formally, a users u can use permission p if:

$$\exists r : [(u, r) \in UR] \wedge [(r, p) \in PA]$$

Further information on RBAC$_0$ policies can be found in [22].

3.2 Cryptographic RBAC$_0$

In the cryptographic RBAC$_0$ policy, each user u and each role r is provided with a pair of public-private keys, respectively indicated as $(\mathbf{k}_U^{\mathbf{pri}}, \mathbf{k}_U^{\mathbf{pub}})$ and $(\mathbf{k}_R^{\mathbf{pri}}, \mathbf{k}_R^{\mathbf{pub}})$. How a user obtains his/her keys depends on the chosen public-private key algorithm of the Hybrid Encryption cryptosystem. In [15] the authors proposed two different asymmetric algorithms: Public Key Infrastructure (PKI) and Identity Based Encryption - Identity Based Signature (IBE - IBS). In the first case, user's keys are self-generated, while in the second case the private key is derived from a Master Secret Key known only by the administrator. The public key is a string uniquely associated with the user (e.g., his/her email). The key pair of the role, instead, is always generated by the administrator. The permissions involve only files as resources and `Read` and `Write` as actions. Actually, as detailed in Section 3.2.1, the `Write` action includes the ability to access the content of the file. Therefore, the permissions that can be distributed by the administrator are actually `Read` and `Read-Write`. Each file is encrypted with a different symmetric key $\mathbf{k}^{\mathbf{sym}}$. The RBAC$_0$ policy is publicly accessible and, as shown in Figure 3.1 (page 26), it is encoded through the following metadata:

- **Lists:** these are three lists containing all the entities present in the RBAC$_0$ policy. The first one corresponds to the U set and is therefore related to the users. It contains their IDs along with their public keys. Note that there may be sensitive information contained in the users' list, for instance in case the chosen public-private key algorithm is IBE - IBS and users' emails are the public keys. The second list corresponds to the R set and collects the roles by considering their IDs and public keys. The last list is related to the files and contains their IDs.

- **Tuples:** they encode the UR and PA sets of assignments. To ensure the integrity of the tuples against malicious or accidental modifications, they are digitally signed with the private key of the user or role that legitimately created or modified them. Usually, it is the administrator of the policy that creates and modifies the policy and therefore signs the tuples. Every time a user accesses the tuples, he/she validates the digital signature. For simplicity, we omit the signature from the representation of the tuples. Figure 3.1 (page 26) reports the three kinds of tuples related to the RBAC$_0$ policy:

  * *Role Tuples*: defined by the `RK` flag, these represent the set of assignments UR that defines, for each user, the roles he/she is assigned to. Hence, a role tuple is composed by the user's and role's IDs (u, r), together with the keypair of the role encrypted with the user's public key $\mathbf{k}_U^{\mathbf{pub}}$, resulting in $(\{\mathbf{k}_R^{\mathbf{pri}}\}_{\mathbf{k}_U^{\mathbf{pub}}}, \{\mathbf{k}_R^{\mathbf{pub}}\}_{\mathbf{k}_U^{\mathbf{pub}}})$. Therefore, a user is able to decrypt the keys of the roles he/she is assigned to.

  * *Permission Tuples*: defined by the `FK` flag, these encode the set of assignments PA. Each tuple specifies which role has access to which file by specifying the ID of the file (f) and the granted action (op). Also, each tuple contains the symmetric key used to encrypt the file encrypted with the role's public key $\mathbf{k}_R^{\mathbf{pub}}$, resulting in $\{\mathbf{k}^{\mathbf{sym}}\}_{\mathbf{k}_R^{\mathbf{pub}}}$. Therefore, with the private key of the role $\mathbf{k}_R^{\mathbf{pri}}$, it is possible to decrypt the symmetric key $\mathbf{k}^{\mathbf{sym}}$.

  * *File Tuples*: defined by the `F` flag, these tuples collect together the encrypted files $\{File\}_{\mathbf{k}^{\mathbf{sym}}}$ along with their IDs (f).

$$\langle \mathsf{RK}, u, r, \{\mathbf{k}_R^{\mathbf{pri}}\}_{\mathbf{k}_U^{\mathbf{pub}}}, \{\mathbf{k}_R^{\mathbf{pub}}\}_{\mathbf{k}_U^{\mathbf{pub}}} \rangle$$
$$\langle \mathsf{FK}, r, f, op, \{\mathbf{k}^{\mathbf{sym}}\}_{\mathbf{k}_R^{\mathbf{pub}}} \rangle$$
$$\langle \mathsf{F}, f, \{File\}_{\mathbf{k}^{\mathbf{sym}}} \rangle$$

Figure 3.1: Tuples Presented in [15]

3.2.1 Read and Write Actions

Suppose the presence of a user, Bob, who is in possession of public-private keys indicated as $(\mathbf{k}_B^{\mathbf{pri}}, \mathbf{k}_B^{\mathbf{pub}})$. Assume also the presence of a role, Professor, having private-public keys $(\mathbf{k}_P^{\mathbf{pri}}, \mathbf{k}_P^{\mathbf{pub}})$ as well. Bob is assigned to the role of Professor. Finally, suppose the presence of a file, named *Exam*, over which Professors have `Read` and `Write` permissions. The setup phase for the enforcing of this policy consists of these steps performed by the administrator:

- The file *Exam* is encrypted with a symmetric key $\mathbf{k}^{\mathbf{sym}}$ generated by the administrator, resulting in $\{Exam\}_{\mathbf{k}^{\mathbf{sym}}}$. The plain-text version of the file is deleted.
- The symmetric key $\mathbf{k}^{\mathbf{sym}}$ is encrypted with $\mathbf{k}_P^{\mathbf{pub}}$, resulting in $\{\mathbf{k}^{\mathbf{sym}}\}_{\mathbf{k}_P^{\mathbf{pub}}}$. The plain-text $\mathbf{k}^{\mathbf{sym}}$ is deleted.
- The Professor's private key $\mathbf{k}_P^{\mathbf{pri}}$ is encrypted with $\mathbf{k}_B^{\mathbf{pub}}$, resulting in $\{\mathbf{k}_P^{\mathbf{pri}}\}_{\mathbf{k}_B^{\mathbf{pub}}}$. The plain-text $\mathbf{k}_P^{\mathbf{pri}}$ is deleted.

Figure 3.2 (page 26) represents the scenario just described. As depicted, both the user Bob and the role Professor have a public-private keypair, while the file *Exam* is encrypted with the symmetric key.

Figure 3.2: Users, Roles, and Files Keys Setup

Read File Suppose that Bob wants to read the file *Exam*. Like every other user, he can freely download the encrypted file from the Cloud. To access the content of the file, Bob performs the following operations:

1. Bob uses his private key $\mathbf{k}_B^{\mathbf{pri}}$ to decrypt the Professor's encrypted private key $\{\mathbf{k}_P^{\mathbf{pri}}\}_{\mathbf{k}_B^{\mathbf{pub}}}$, obtaining $\mathbf{k}_P^{\mathbf{pri}}$. In this step, Bob is assuming the role Professor as identity.
2. Bob (as Professor) uses $\mathbf{k}_P^{\mathbf{pri}}$ to decrypt the encrypted symmetric key $\{\mathbf{k}^{\mathbf{sym}}\}_{\mathbf{k}_P^{\mathbf{pub}}}$, obtaining $\mathbf{k}^{\mathbf{sym}}$.

3. Bob (as Professor) uses $\mathbf{k}^{\mathbf{sym}}$ to decrypt the content of the encrypted file $\{Exam\}_{\mathbf{k}^{\mathbf{sym}}}$, obtaining the plain text of *Exam*.

The `Read` action on a file is straightforward since the there is no need for other entities to take part in the communication, as this action does not imply the modification of data. Figure 3.3 (page 27) represents the sequence of Bob's operations.

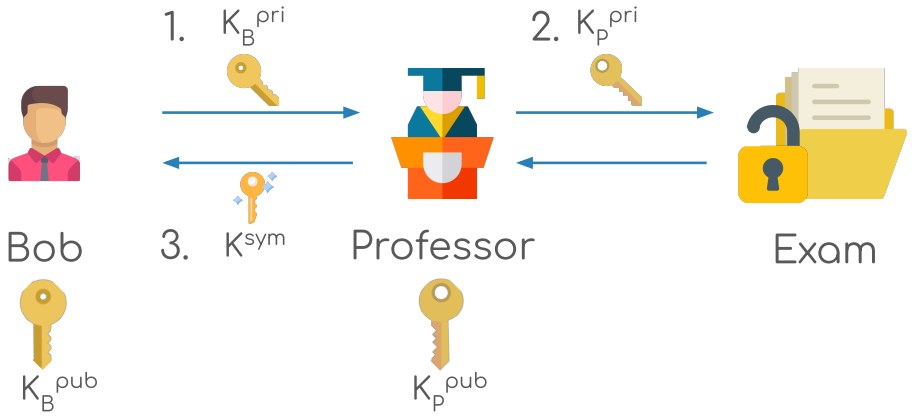


Figure 3.3: Keys Exchange in Read Action

Suppose now that another user, named Carl, wants to access the content of the file *Exam*. However, Carl is not assigned to the role of Professor. Therefore, when he downloads the encrypted file from the Cloud, he cannot access its content. In fact, the only way to decrypt $\{\mathbf{k}^{\mathbf{sym}}\}_{\mathbf{k}_P^{\mathbf{pub}}}$ is to possess the Professor's private key $\mathbf{k}_P^{\mathbf{pri}}$, and the only way to obtain it is by decrypting $\{\mathbf{k}_P^{\mathbf{pri}}\}_{\mathbf{k}_B^{\mathbf{pub}}}$. Since Carl does not possess Bob's private key $\mathbf{k}_B^{\mathbf{pri}}$, he cannot decrypt the key of the role. Therefore, the encrypted content of the file *Exam* remains inaccessible.

Write File Suppose that Bob wants to update the content of the file *Exam*. To explain how this works, we have to introduce another entity, that is a partially trusted RM placed between users and files. In this case, the RM has to checks that the Bob's has actually a valid `Write` permission and that the digital signature of the new file is valid. Therefore, to modify the content of a file under the cryptographic RBAC_0 scheme, Bob performs the following operations:

1. Bob uses his private key $\mathbf{k}_B^{\mathbf{pri}}$ to decrypt the Professor's encrypted private key $\{\mathbf{k}_P^{\mathbf{pri}}\}_{\mathbf{k}_B^{\mathbf{pub}}}$, obtaining in $\mathbf{k}_P^{\mathbf{pri}}$ In this step, Bob is assuming the role Professor as identity.
2. Bob (as Professor) uses $\mathbf{k}_P^{\mathbf{pri}}$ to decrypt the encrypted symmetric key $\{\mathbf{k}^{\mathbf{sym}}\}_{\mathbf{k}_P^{\mathbf{pub}}}$, obtaining $\mathbf{k}^{\mathbf{sym}}$.
3. Bob (as Professor) encrypts the new version of the file $Exam^2$ with $\mathbf{k}^{\mathbf{sym}}$, resulting in $\{Exam^2\}_{\mathbf{k}^{\mathbf{sym}}}$, and digitally sign the new file with his private key $\mathbf{k}_B^{\mathbf{pri}}$.
4. Bob (as Professor) sends the new version of the encrypted file $\{Exam^2\}_{\mathbf{k}^{\mathbf{sym}}}$ to the RM.
5. The RM checks that the role Professor has `Write` permission over the file *Exam* and the validity of the digital signature. If the outcome is positive, it allows the request.

With respect to the `Read` action, here there is also the participation of the RM entity. This is necessary since this action involves the modification of data in the Cloud, and this has to be

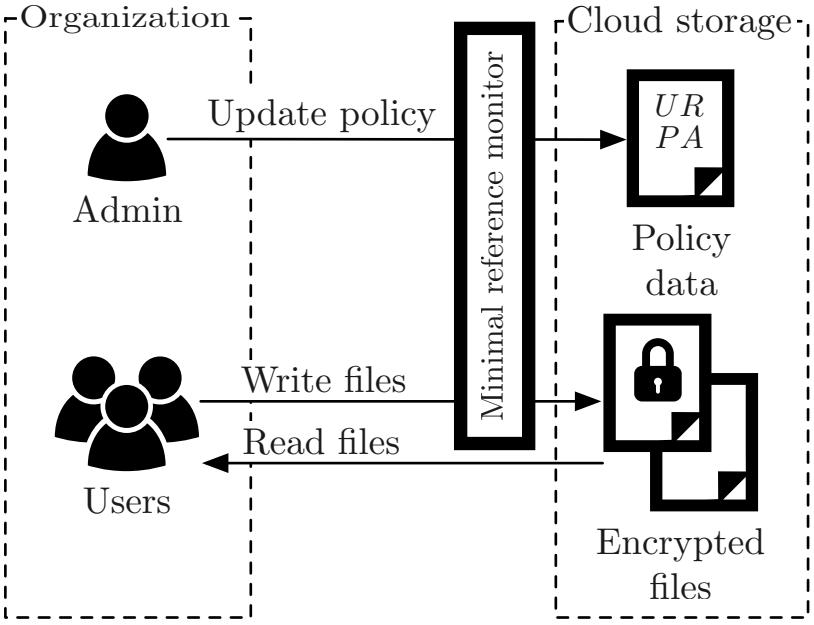


Figure 3.4: Architecture Proposed in [15]

checked against the policy. Every modification of files or of the policy has to be approved by the RM, as shown in Figure 3.4 (page 28).

Suppose now that the user Carl is assigned to the role Student, that has only `Read` permission over the file *Exam*. In this case, Carl can access the symmetric key $\mathbf{k}^{\mathbf{sym}}$ and therefore can produce a valid ciphertext for an eventual new version of the file. However, when Carl sends the new file to the RM, this rejects the modification because the Student role does not have `Write` permission over the file *Exam*.

3.3 Entities Involved in Cryptographic $RBAC_0$

From the description in [15], it is possible to identify five entities actively involved in the functioning of the scheme, Each assigned specific operations. A high-level view of the entities along with the related operations is below presented and reported in Figure 3.5 (page 29).

Administrator. It represents a single entity having full control over the $RBAC_0$ policy. In a company, this entity reflects the person in charge of handling the whole policy. The administrator can add users and roles and create assignments between them. Also, he can distribute permissions to roles over files. The only permissions that can be granted are `Read` and `Read-Write` over files. Note that the administrator is a fully trusted entity but, Since it has the capability of modifying the policy, this trust is not a weakness [15].

Clients. They are the users of the cryptographic $RBAC_0$ policy. In a company, they reflect the employees. The cryptographic $RBAC_0$ policy regulates their accesses to files in the Cloud. Clients have only three operations through which interact with the files:

- `Add`: clients can add new files. Note that there is no need to grant any permission for executing this action.
- `Read`: given a file, this action allows clients to read its content. Clients can read only files they have been given access to by the administrator.

- `Write`: given a file, this action allows clients to modify its content. Clients can write only on files they have been given access to by the administrator.

Reference Monitor. This entity is logically positioned between the clients and the Cloud. Its duty is to check the integrity of every modification on files or on the policy. In the first case, the RM checks that the client requesting to modify the file has actually valid permission over it and that the digital signature of the new file is valid. In the second case, the RM checks the digital signature created by the administrator on the new tuples.

CryptoAC. This entity implements the algorithm supporting the cryptographic RBAC_0 scheme. As such, its duty is to transform clients' and administrator's requests into the sequence of cryptographic operations necessary to accomplish them. For instance, whenever a client requests to read a file, he/she invokes CryptoAC that, if possible, will decrypt the file and return it. Because CryptoAC is managing private keys and sensitive information (e.g., the content of the files), it cannot run on the CSP.

CSP. Encrypted files are sent by the RM to the CSP that is accounted for a secure and reliable storage. However, it is not a fully trusted entity but an Honest but Curious one. This implies that a company can trust it to execute assigned instructions but the CSP will try to access and read the content of stored files.

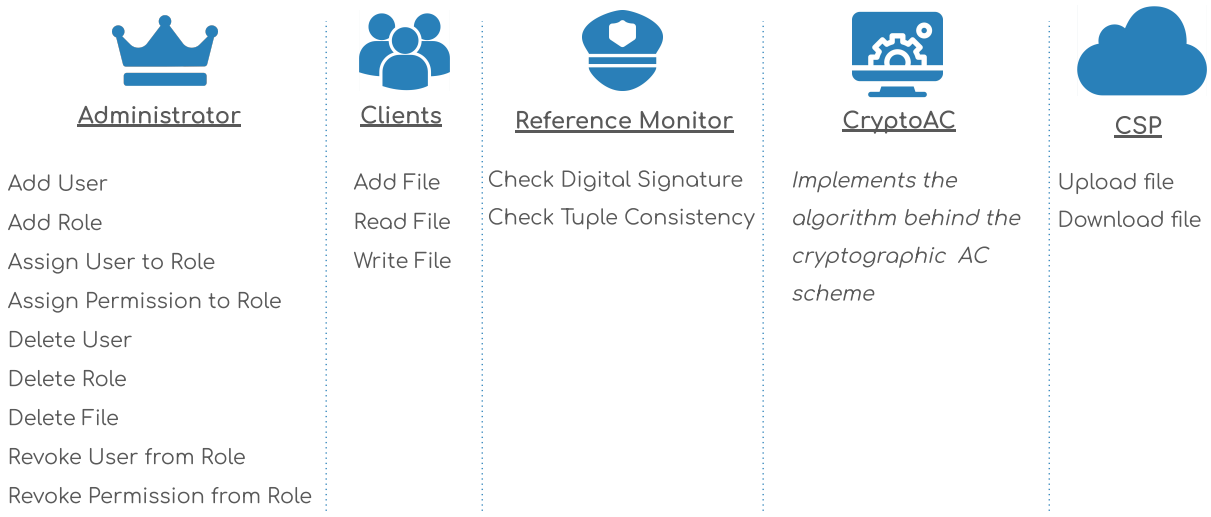


Figure 3.5: Entities and Related Operations in [15]

3.3.1 Dynamic Behaviour of the Policy

One of the most important properties of the scheme proposed in [15] is the modification at runtime of the policy. In particular, this means that entities can be deleted and permissions can be revoked. Each role and each file is assigned a version number to maintain track of the status of the policy. While the `Write` action is controlled by the RM, the `Read` one occurs directly. Being so, when the `Read` permission is revoked from a role, there is the need of re-encrypting the involved file under a new symmetric key. Even worse, when a user is removed from a role, all files that the role can access to have to be re-encrypted. This prevents the user from accessing old files with cached keys. Also, the keys of the role are changed so that the user cannot obtain the new symmetric

keys of the files. Even though the authors in [15] exploits mechanisms like lazy re-encryption, this is a cumbersome procedure that is likely to produce prohibitive overheads, limiting the usability of the scheme. However, this consideration about efficiency also applies to every other dynamic cryptographic AC scheme and not only the one proposed in [15].

3.4 Cryptographic AC Scheme Choice Motivations

As presented in Chapter 2, many researchers proposed different schemes for enforcing cryptographic AC policies in the Cloud. Still, many focused only on high-level and the abstract properties of their scheme, often not considering an actual implementation. This implies that they omit important aspects like general efficiency, revocation of keys, delegated re-encryption and dynamic changes in the policy. Instead, Garrison et al. [15] specifically addressed these issues to be closer to a real-world use case. In detail, we chose to consider their work as the starting point of ours because:

- They developed an algorithm for enforcing a simple but expressive policy (RBAC_0) that is widely adopted and therefore can be exploited in a concrete implementation.
- They added a dynamic behaviour to the policy, an essential property for a realistic use case.
- They took into account the efficiency of the cryptographic RBAC_0 scheme. By choosing a simple policy, this scheme enhances the speed of execution of the several cryptographic related operations. Furthermore, the authors discussed the advantages of lazy encryption against re-encryption when revoking users or permissions from the policy. Also, their cryptographic scheme is hybrid so to exploit symmetric encryption for files, an approach many times faster than asymmetric techniques.
- They conducted a complete analysis of the issues that were instead scarcely addressed by other similar works. In particular, they discussed assumptions on the use case in which their scheme is designed, like the trust that can be given to the CSP. The complete list can be found in Section 3.5.
- They suggested two possible architectures for what concerns the relationships of the involved entities. In the first, the RM runs on the CSP. Clients have direct access to files through the provided APIs. The `Read` action is enforced cryptographically, while the `Write` one is to be checked by the RM. The second architecture is instead more structured. A proxy, deployed by the company, mediates all the communications between the users and the CSP. Since the proxy is supposed to be deployed by the company itself, it can be considered trusted and secure. In this case, both the `Read` and the `Write` actions can be directly enforced by the proxy itself.
- they presented pseudocode (see Appendix A) for an eventual implementation.
- They provided a formal demonstration of the soundness of their algorithm through the parametrized expressiveness framework [35] and performed a simulation on the efficiency of their scheme with Markov chains.

3.5 Abstract Assumptions and Limitations

In this section, we discuss assumptions and limitations related to the use case described in [15]. They are essential to understand the background in which the cryptographic RBAC_0 scheme is designed.

They are either explicitly stated or inferred from the presented use case. We detail the reasons why they were formulated in the first place and their implications on the use case. Note that these are related only to the abstract scheme, hence they do not take into account any concrete implementation. In Table 3.1 (page 33) we report the summary of such assumptions and limitations.

3.5.1 Abstract Assumptions

Being an abstract formulation, several assumptions were made on the presented use case and the related threat model. Some are bounded to the entities that are actively involved in the functioning of the cryptographic RBAC_0 scheme. Some are derived from the specific and peculiar contributions of the paper (e.g., the dynamicity of the scheme). At last, others were formulated to avoid too much complexity in discussing the cryptographic RBAC_0 scheme and to postpone low-level details.

- **A1 - CSP can Execute Code**: the CSP has the ability to execute code, since the RM can also run on the CSP.
- **A2 - CSP Trusted as Storage Solution**: this states that the CSP is trusted to safely store files that the clients uploads. The CSP has to ensure the availability of these files by serving them to authorized clients and blocking unauthorized accesses through the RM. Together with files, the CSP can store the metadata encoding the RBAC_0 policy, even though with a different service (e.g. a relational DB). This is a fundamental assumption that upholds the cryptographic RBAC_0 scheme. If there was not a CSP with the ability to store files, the use case would not exist.
- **A3 - Consider only one CSP**: given a company with a AC policy to enforce over a set of files, it is assumed that all files are stored in one and one specific CSP only. This consideration is actually close to the way usually companies employ CSPs, especially for what concerns outsourcing the storage of their data. In fact, having a company using more than one CSP for saving the same files at the same time is a not realistic scenario [15]. Besides files, also all cryptographic metadata can be assumed to be stored together and not distributed in several CSPs. Note that metadata can also be stored offline within the network of the company.
- **A4 - No Spurious Downloads from Users**: this assumption addresses the problem of a client overwhelming the CSP with spurious file downloads. Being metadata publicly accessible, all users have access to names of all files. Since the `Read` action occurs directly, every user can download every file he/she wants even though not able to decrypt it. The authors [15] say that this problem is "easily addressed by using unguessable [...] file names".
- **A5 - Not considering Concurrency**: this issue is largely independent from the cryptographic RBAC_0 scheme. In fact, concurrency is a problem that has to be tackled by any implementation interacting with an external database or storage. In the use case described in [15] two clients could simultaneously write on a file, or a client could start a `Write` action in the exact moment his/her permission is being revoked, or the administrator could execute two revoke operations simultaneously. Depending on how the architecture is structured, this may lead the scheme in an inconsistent state.
- **A6 - CSP is Curious**: this is a corollary of the presented use case: the CSP will try to read the content of the files uploaded by the company. There are situations in which this conduct may not be acceptable, like when sensitive data are involved. Note that the fact that the CSP is

not a fully trusted entity is one of the fundamental assumptions that composes the ground for cryptographic AC and gives meaning to the whole scheme.

- **A7 - Digital Signatures on Tuples**: the tuples encoding the policy are assumed to be digitally signed by the users or roles that either created or modified them. As such, it is possible to verify their validity of the tuples and detect unauthorized changes in the $RBAC_0$ policy.
- **A8 - Secure Channels for Communications**: every communication happening between the five entities is assumed to be secure and pairwise-authenticated.

3.5.2 Abstract Limitations

Although several studies have been conducted on the topic of cryptographic AC, there are still many limitations that hinder a concrete implementation. Mainly, these are implicitly derived from the use of cryptography itself. Besides the base complexity of an AC policy, there is also the overhead of cryptographic operations and the management of the dynamicity of the policy. As explained in Chapter 2, there is a clear tradeoff between the overall efficiency of the scheme and the expressiveness of the related policy. The more complex the policy to enforce, the higher the computational load of the scheme. Since the authors in [15] decided to prioritize efficiency, the capability of the AC scheme is consequently reduced in these aspects:

- **L1 - Adoptable Policies**: cryptographic AC schemes tend to be computationally expensive, especially when dynamic. To get as close as possible to a feasible use case, the authors decided to use enforce $RBAC_0$ policy only. The result is a simple policy with a limited expressivity with respect to the users, for which only their belonging to a role decide whether or not to grant access to a specific file.
- **L2 - Permissions Assignable**: usually AC policies cover various kind of actions on a set of several kinds of resources. For instance, a user can be given the possibility to create a folder, delegate authorizations, create other users or even gain administrator privileges. Unfortunately, when considering this particular cryptographic $RBAC_0$ scheme, the only kind of permissions assignable are `Read` and `Read-Write`. Moreover, the only resources over which it is possible to give permissions are files.
- **L3 - High Computational Costs**: even with these simplifications, the adoption of the scheme still implies a heavy computational burden. Besides the demonstration of the soundness of the algorithm, the authors conducted a simulation on real-world datasets: the outcome is that, even in a minimally-dynamic use case, the scheme is likely to produce too much overhead for a feasible deployment. This result hints that an AC policy based on cryptography only may not be sustainable.

In the next Chapters, we describe further requirements and assumptions that we developed on the use case. We sum up these two sets to define the guidelines for the designing of the pragmatic approach for the deployment of the cryptographic $RBAC_0$ scheme.

Abstract Assumption	Description
A1	CSP can Execute Code
A2	CSP Trusted as Storage Solution
A3	Consider only one CSP
A4	No Spurious Downloads from Users
A5	Not considering Concurrency
A6	CSP is Curious
A7	Digital Signatures on Tuples
A8	Secure Channels for Communications

Abstract Limitation	Description
L1	Adoptable Policies
L2	Permissions Assignable
L3	High Computational Costs

Table 3.1: Abstract Assumptions and Limitations from [15]

Chapter 4

Concrete Requirements and Assumptions

There are still some important aspects not considered in [15] that are of great importance when considering a concrete implementation, like keys management. It is plain that clients' private keys cannot be stored in the CSP. In fact, this would be against assumption A5, since then the CSP would be able to decrypt files and read their content, even though indirectly. Still, it is not clear which entity should manage and store the private keys. Another point is that privacy was not considered in the design of the cryptographic RBAC_0 scheme. Clients interact with the database to retrieve public keys, check their assignments to the roles and files and eventually validate the signatures of the tuples. Unfortunately, limitations on how metadata are accessed are not defined and instead lists and tuples are said to be publicly accessible. By design, clients have access to the complete list of entities currently present in the RBAC_0 policy. Moreover, they have also knowledge of the assignments users-roles and roles-files of other clients. Disclosing the full policy might not be acceptable in a company, so there is the need to slightly change the way clients access metadata so to protect such sensitive information.

In this Chapter, we first define a set of high-level design properties. We focus on these so to guarantee a pragmatic approach to the deployment of the scheme. From these properties, we derive concrete requirements for our architecture. Moreover, we discuss in detail the assumptions presented in Chapter 3. While keeping some for our architecture, we transform other assumptions into actual requirements. In fact, even though some concepts might have been formulated as assumptions at the abstract level, they become requirements at the implementation level (e.g. A8 Secure Channels for Communications). These add up to the one previously described and together compose the list of requirements that our architecture must respect. We summarize all assumptions and requirements along with their origin in Figure 4.1 (page 36).

4.1 High-Level Design Properties

Since the contribution of this thesis is a pragmatic approach to deploy the cryptographic RBAC_0 scheme. We want to cover as many use cases as possible. This implies taking into account several options for the interaction among the different entities, being able to support multiple CSPs and, most important, letting the company the possibility to customize the architecture to best fit its needs. To achieve all of this, we choose to highlight and discuss these four high-level properties:

- **Flexibility**: the company should be able to adapt our architecture to its needs as much as possible. This includes the ability to modify the information flow and relationships in the architecture in terms of interactions and communications, physical location of the metadata and of the involved entities as well.

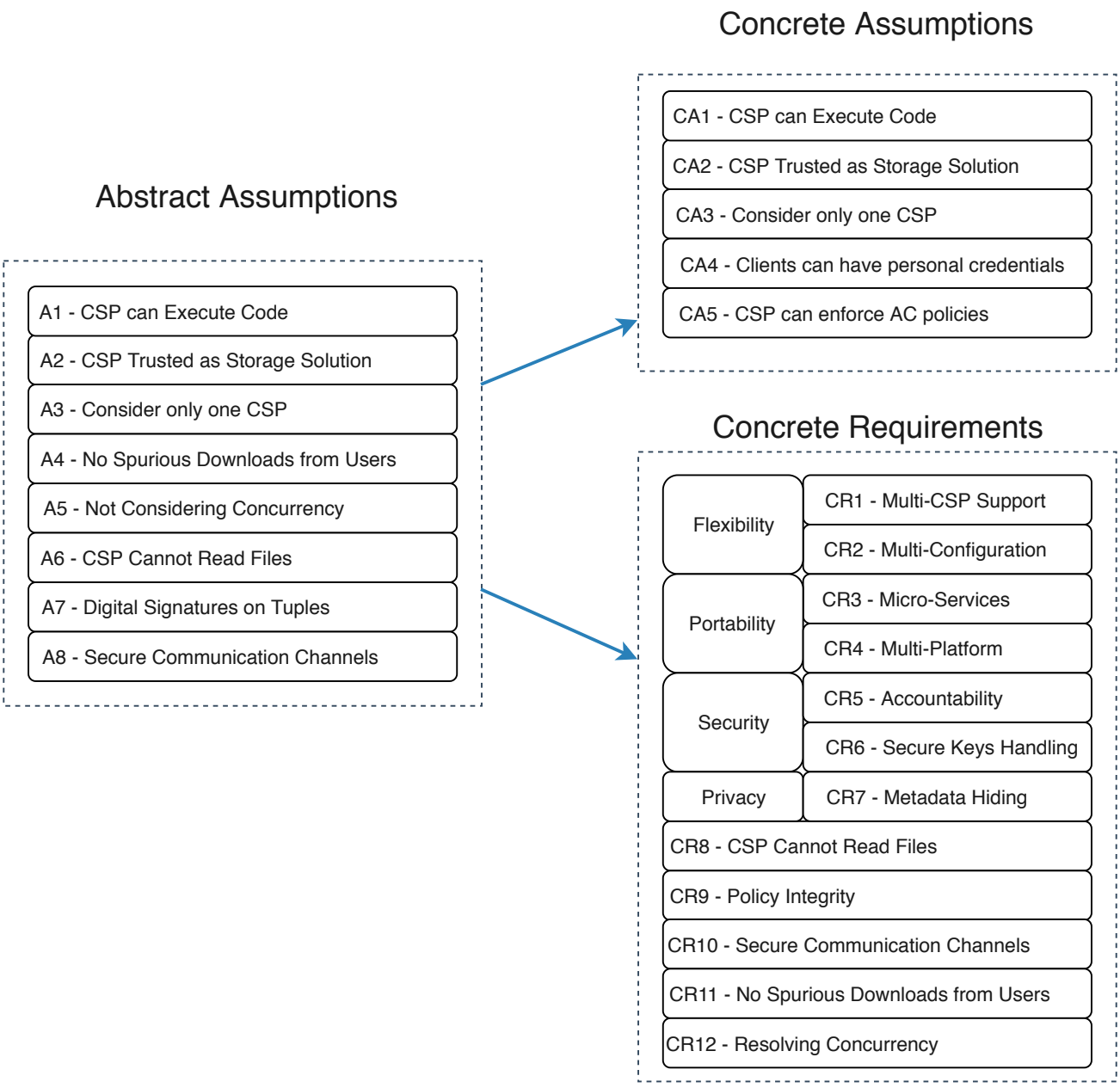


Figure 4.1: Derivation of Requirements and Assumptions

For instance, the architecture should be agnostic about the specific CSP employed: since each company may use a different one, it is required to support multiple CSPs. This is emphasized by the fact that their capabilities may differ, so the design of the architecture cannot rely on peculiar services offered by a single of few CSPs. On the other hand, to avoid too much complexity and exploit advanced functionalities, our architecture should also take advantage of common features among CSPs.

- **Portability**: the design of the architecture should guarantee as much portability as possible. In this way, we ensure easiness in the deployment and reduce the effort by the company during the setup phase. Portability implies also an easier management and maintenance of the architecture, for instance through a modular design.

  A concrete point to consider for a pragmatic deployment is that the architecture should not prevent the possibility to run on all Operative Systems (OSs) and devices (e.g. computer but

also smartphones).

- **Security**: since the use case is about storing sensitive data in the Cloud, an obvious property is the security and the protection of communications. This includes the ability to share information between entities so that no third-party can access or read the exchanged data. Also, the design of the architecture should ensure that sensitive information (e.g., private keys) flows across the network as rarely as possible.

  For instance, this property comprehends a secure handling of cryptographic PKI or IBE - IBS keys and also protections against general malicious attacks like SQL injections.

- **Privacy**: this property is not addressed in [15] since the cryptographic $RBAC_0$ scheme is abstract and focuses on soundness and efficiency. However, in a company it is fundamental to preserve the confidentiality of information, considering both internal and external threats. In fact, if files and especially metadata are not properly protected, they could reveal sensitive information.

  In particular, this property implies a sensible management of metadata related to the cryptographic $RBAC_0$ scheme, like the lists of users or the relationship users-roles. Especially, this aspect is emphasized when sensitive information is involved (e.g., users' personal data for IBE - IBS algorithm).

There are also other important properties like **testability** and **usability**, crucial for a concrete implementation. However, they are not part of the discussion since they relate more to classic software engineering issues and challenges rather than to cryptographic AC. Therefore, they are not specific to our use case and addressing them would not produce any real contribution.

4.2 Concrete Requirements From High-Level Design Properties

We proceed to describe the requirements we formulated taking into account the four high-level properties just described in Section 4.1. By respecting these requirements we ensure the presence of these properties. Since these push the design toward a feasible and realistic architecture, we make it more appealing for a real deployment.

- **CR1 - Multi-CSP Support**: since our architecture is supposed to be deployed by different companies, it is essential to design it to support more than one CSP. This requirement does not contrast assumption A2 (Only one CSP). In fact, A2 was placed to avoid synchronization issues among CSP used simultaneously under the same $RBAC_0$ policy. On the contrary, here we argue that the architecture should be capable of interacting with different CSPs but only one at a time. In this way, two companies employing different CSPs could both independently adopt our architecture. Since the services offered by CSPs may change from one another, the architecture must be designed so that it does not assume the presence of a peculiar service so to be independent from the chosen CSP.

- **CR2 - Multi-Configuration**: the architecture should operate in different configurations. This is due to the impossibility to know where and in what use case it will be deployed. Since various companies may have divergent needs, each one of these configurations should favour a different property to meet the specific needs of the company. Also, it should be possible to easily decide which one to adopt and to leave the final choice to the company itself.

- **CR3 - Micro-Services**: given again the unknown environment, the architecture should be modular by working through micro-services to be deployed effortlessly. At the implementation level, this translates in a modularization of the software components embedding these functionalities. This requirement also increases the scalability of the service.
- **CR4 - Multi-Platform**: a common feature of portable architectures is the ability to deploy them across several platforms to cover the heterogeneity of OS and devices available.
- **CR5 - Accountability**: in a company, it may be important to track the operations of the employees and have a way to keep them accountable for their behaviour. Even though authorized, employees can apport modifications and changes to files that the administrator may want to trace. Since tuples are digitally signed by users and roles, it is partially possible to keep them liable for their operations. However, whenever a `Write` action occurs in [15], the resulting tuple is signed by a role, and not by the user. Hence, it is not feasible to go back to the user that changed a file but only to the role that accessed it.
- **CR6 - Secure keys Handling**: a crucial requirement for assuring the consistency of the whole scheme is the safe handling of the PKI or IBE - IBS keys. In fact, the integrity of the entire scheme depends on this. For this reason, cryptographic keys should be securely stored and exchanged across the network as little as possible, even though through a secure channel. Of course, private keys cannot be stored in the Cloud.
- **CR7 - Metadata Hiding**: lists and tuples are defined as freely accessible in [15]. Instead, we introduce the opposite requirement: users should not know what other users are in the $RBAC_0$ policy. Also, they should not know what other roles or files are present besides the ones assigned to them. Finally, users should not know the relationships between user-roles and roles-files expect their own. Such a requirement has to be valid for every CSP may be used.

In this discussion, we ignore other kinds of security aspects like resilience against SQL injections or other malicious attacks, because too much general and not bounded to our specific use case.

4.3 Abstract Assumptions Analysis

We now analyze the abstract assumptions illustrated in Section 3.5. For each one, we further discuss whether to keep it or to transform it into a concrete requirement for our architecture. In the former case, we show that the abstract assumption is reasonable and useful to our architecture, mutating it to a concrete assumption. In the latter case, we present how the abstract assumption is transformed into a concrete requirement.

4.3.1 Concrete Assumptions

There are two different kinds of concrete assumptions: the one retained from the previous discussion in Chapter 3 and the one formulated by us.

Assumption Retained Related to the CSP, we conserve the assumption A1 as **CA1: CSP can Execute Code**. This is fundamental to allow the RM to be placed in the Cloud. If this were not the case, we would have to find another method to ensure the integrity of `Write` actions. For instance, we could introduce the versioning of the files. Users could always upload a new version of a file.

Then, it is the duty of other users to validate the new version by checking the AC policy. However, this would have a remarkable impact on efficiency and costs. First, users could spend time validating several spurious versions of a file before reaching the valid one. Then, the storage of many copies of the same files would lead to noticeable monetary costs to the company. Moreover, there would be the need of another entity periodically removing spurious versions of files from the Cloud. Another option would be to impose a proxy to mediate users' operations toward the CSP, but this would stiffen the architecture. Therefore, for all the issues just presented, we conserve this assumption. We argue that this assumption is reasonable since such a service is provided by many of the major CSPs. For instance, AWS has the so-called "Lambda" functions [36], while Google and IBM have more generic "Cloud Functions" [37, 38].

Concerning the CSP, we conserve also assumption A2 as **CA2: CSP Trusted as Storage Solution**, since it allows us to place trust in the CSP and therefore let it store files and run the RM. It this was not the case, we would have to exclude the CSP from the set of involved entities. Doing so, we would make cryptographic AC scheme useless in the first place.

We conserve A3 as **CA3: Consider only one CSP** to simplify the architecture and avoid synchronization issues, anyway not related to the cryptographic $RBAC_0$ scheme. Furthermore, note that having multiple copies of the same file distributed across several CSPs would just create monetary costs and performance deterioration, not bringing any concrete advantage.

Our Assumptions In addition to these, we formulate two more assumptions. The first one is **CA4: Clients can have personal credentials**. It states that each client is given personal credentials for authentication toward the CSP. This is needed to let clients have a direct connection without accessing through a proxy. Moreover, we can increase overall accountability, by being able to link each operation to the client that performed it. Clients are securely given such credentials either handed from the administrator or sent by the CSP itself (e.g., through e-mails). The second assumption is **CA5 - CSP can enforce AC Policies**: this means that the CSP offers the capability of configuring AC policies for screening clients' actions besides the `Write` action over files through the RM. In this way, it is possible to limit the operations clients can perform when directly connected to the CSP. For instance, we can restrict the available operations only to the upload and download of files. Hence, we can remove every administrative permission not needed for the functioning of the cryptographic $RBAC_0$ scheme, like the elimination of folders or the creation and assignment of permissions to other clients. This assumption is reasonable since every major CSP offers a service for IAM, like AWS [9] and Azure [8].

4.3.2 Concrete Requirements From Abstract Assumptions

We now describe the list of concrete requirements derived from the abstract assumptions not retained. While some abstract assumptions are basic for the functioning of the scheme, others were formulated just to avoid complexity at an abstract level. Considering a concrete implementation, they have to be addressed. For instance, assumption A6 (*CSP Cannot Read Files*) is to be mutated into a requirement. In fact, this is the ground notion for which the cryptographic AC scheme is used for. Therefore, we introduce the concrete requirement **CR8: CSP Cannot Read Files**, stating that the CSP should not be able to read the content of the files uploaded by the clients in any way. This comprehends:

- **Direct or Passive Access** - the CSP is given the possibility to read the content of a file without

any further operation but reading the file itself. For instance, consider the case in which files are uploaded and stored as plain-text. Another example is the delegation of the encryption operations to the CSP. In this way, files would be stored encrypted, but the content would have been revealed to the CSP.

- **Indirect or Active Access** - the CSP is willing to actively operate to decrypt files so to read their content. This may occur if the chosen architecture expects users' private keys to be stored unencrypted in the Cloud. The same would happen if the CryptoAC entity runs on the CSP. The CSP would be able to obtain the user's private key and decrypt the content of the file. Note that also the RM entity cannot have access to the content of the files, since it may run Cloud-side.

From assumption A7 (*Digital Signatures*) we can infer another requirement **CR9: Policy Integrity**. Users can perform two main actions over the stored files. The `Read` (i) action is enforced cryptographically, so it is not feasible for any entity to change the policy so to gain access to the content of a file. Indeed, this entity would need the key for decrypting the private key of a role that has permissions over the file, so to decrypt the symmetric key to access its content. Since this cannot happen through modifications of the tuples, in no possible way the policy can be altered to leak the content of a file. Instead, the `Write` (ii) action is enforced by the RM that checks the presence of the specific RK and FK tuples and, if these exist, overwrites the file. Thus, a user with the capability of altering or adding tuples would be able to replace all files stored in the Cloud with arbitrary content, although not being able to actually read the old content. Therefore, to preserve the integrity of the $RBAC_0$ policy against accidental or malicious modifications, all tuples are required to be digitally signed with the private key of the user or role that created or legitimately modified it. Any information associated with a wrong or missing signature should be immediately rejected. In this way, even though an entity modifies the policy, it would not be able to correctly sign the new or altered tuple. Therefore, the tuple would be rejected by the RM. We note that the CSP does not need to modify the policy to overwrite the content of the files. In fact, it could easily replace all stored files by itself. However, this is against assumption A2.

Another assumption to transform in requirement is A8 (*Secure Communication Channels*) that becomes **CR10: Secure Communication Channels**. This requirement applies to every exchange of data happening between entities. This comprehends every communication toward the CSP related to both files and metadata and also between the clients and the administrator, even though within the secure boundary of the company. In fact, in case the IBE - IBS algorithm is used, users' private keys are derived from a Master Secret Key that only the administrator possesses. Thus there is the need for a secure channel for the communication of this sensitive information from the administrator to the related client. This does not concern the PKI algorithm, where keys are self-generated.

We also turn into a requirement A4 (*No Spurious Downloads from Users*) as **CR11: No Spurious Downloads from Users**, since this could impact the performance and costs of the scheme, and A5 (*Not Considering Concurrency*) as **CR12: Resolving Concurrency** because, even though independent from the cryptographic $RBAC_0$ scheme, it is an issue to be addressed in a concrete implementation.

Chapter 5

Architecture

Having defined the concrete requirements and assumptions, we now discuss the architecture. Remember that we define with **Architecture** the way entities communicate with each other to implement the cryptographic AC scheme, along with the physical location of metadata. Given that the implementation might have to deal with several other use cases beside the one presented in Chapter 3, we analyze the relationships among the involved entities under many points of view to check every possible interaction. As a result, we present two main degrees of freedom regarding the location of metadata and how entities connect to the CSP. The former involves not only the physical location of the metadata but also the responsibility on the integrity and availability of the metadata itself. This has a significant impact on the efficiency of the scheme and even on monetary expenses. The latter is related to which entity should authenticate the clients before allowing them to operate on stored files. This greatly affects the structure of the architecture and it modifies the way the various entities communicate with each other.

First, we discuss these degrees of freedom and argue that, whatever the choice, they do not violate the requirements formulated in Chapter 4. Then, we illustrate which features of the architecture they influence and in what way. In this way, we show that there are several architectures that can correctly implement the cryptographic RBAC_0 scheme we chose. Therefore, our design achieves fine-grained adaptability with respect to the deployment use case. Eventually, we further discuss and develop two specific and opposite architectures. In each of these, we prioritize different features, like the overall efficiency over the amount of control enforceable on users' actions.

5.1 Degrees of Freedom

These two degrees of freedom influence the interactions among the entities and the structure of the architecture but they do not mine the soundness of the scheme. Therefore, different combinations always yield valid architectures, even though with peculiar pros and cons. Some choices may be forced because of other factors unrelated to the architecture itself (e.g., rules or preferences within the company). We now present these degrees of freedom along with an informal reasoning of how they either do not violate or do not affect the requirements previously formulated.

Storage Location of Metadata The first degree of freedom that affects the structure of the architecture is where the lists and tuples are stored. We argue that these can be stored either in the CSP or in a trusted server managed by the company or through a hybrid solution. In all three cases, no requirement is violated. In fact, no special service is requested to the CSP (CR1) for storing the metadata, the

trusted server can be designed as a micro-service (CR3) and accountability is provided with logs written by either the server or the CSP (CR5). The hiding of data in the lists and tuples with respect to the clients is easy to enforce if they are stored in the internal server. If instead metadata are stored in a DB, the DB engine can be used to tune clients' privileges to provide the needed privacy (CR7). Moreover, also the RM can be used to screen the clients' requests. Then, if tuples are stored in the Cloud, the integrity of the policy is guaranteed by digital signatures. Remember that the CSP cannot forge the signatures since it does not have the users' or roles' private keys (CR9). Being so, the CSP cannot read files either (CR8). Other requirements are not influenced by the location of metadata.

Clients' access to the CSP Regardless of where metadata are stored, clients have still to interact with the CSP. In particular, this relates to whenever clients want to perform a **Read** or **Write** operation. Clients can connect either directly or through a dedicated trusted proxy within the boundary of the company. In both cases, note that the RM entity is still able to mediate all the requests, being deployed either in the CSP (supported by assumption CA1) or in the proxy. Therefore, the consistency of clients' actions is ensured. Since the only variation is the addition of a trusted proxy in the middle of communications, all requirements are not affected. This is even more obvious if we think the proxy is just an entity that collects clients' request and forwards them to the CSP.

In case clients have a direct connection with the CSP, an independent choice is the kind of credentials they use for authentication. In fact, they can use either personal credentials (supported by assumption CA4) or temporary credentials generated on the fly by the administrator, acting as a server within the company. The only requirement affected is the accountability of clients' actions (CR5). However, being credentials either personal or unique, this is still satisfiable.

5.2 Features in the Architecture

Both degrees of freedom previously described affect in a different way the architecture. In particular, we identified 10 peculiar features that can be influenced. They are:

- **Deployment Readiness**: this relates to the simplicity in the deployment of the architecture. In practice, it refers to the ability to set up and configure the various entities minimizing the effort put by the company. The fewer steps the company has to perform, the easier the deployment.
- **Keys Location**: the requirement CR6 states that private keys must be securely stored. Still, another point to address is their storage location. In fact, they can be stored in a centralized or distributed fashion. Remember that in no case they can be stored in plain in the Cloud.
- **Scalability**: this refers to the ability of the architecture to efficiently adapt to handle greater workloads. In our use cases, this translates in the possibility to dynamically distribute the usage of bandwidth and computational resources.
- **Workload Management**: considering a static and fixed number of clients, this is the ability of the architecture to bear and manage peaks in the workload. This feature is strictly related to the efficiency of the architecture.
- **Robustness**: this feature measures the ability of the architecture to keep working regardless of possible failures of the entities involved.

- **Control on Metadata**: this is the amount of control the architecture can enforce over lists and tuples, both for regulating accesses and for protecting them.
- **Policy Maintenance and Queries Speed**: the update of the policy by the administrator may cause several re-encryptions of files. Besides, all the related tuples have to be modified. Depending on where tuples and the lists are stored, the speed of such operations may change. This influences also the queries made by the clients whenever reading or writing a file.
- **Monetary Savings**: the more the CSP and its services are used, the more expensive in terms of money it becomes. Besides the storage of files, this feature is affected also by the storage of metadata and the operations we delegate to the CSP. We are not considering costs within the company, but only the ones related to the CSP.
- **Multi-Device**: this feature concerns the ability to access the functionalities offered by the architecture from several and different devices, like smartphones.
- **Control on Accesses**: this related to the amount of control the company can enforce on how employees access to files.

5.3 Degrees of Freedom Applied to the Architecture

We now discuss how the two degrees of freedoms can influence the different features related to the architecture. In particular, for each degree of freedom, we illustrate the possible alternatives and discuss their pros and cons.

5.3.1 Storage Location of Metadata

Remember that the lists of users and roles contain their IDs and public keys, needed to validate the digital signatures of the tuples. The lists of roles and files include version numbers that are crucial to handle revocation. Tuples include data about the relationships users-roles and roles-files, that compose the AC policy itself. Because of this, they are intensively queried whenever a user wants to perform a **Read** or **Write** operation since he/she has to find a role that has permissions over the given file. There are three main options for what concerns the location of metadata:

- **Dedicated Server**: lists and tuples are stored in an always running server managed by the company itself. Users should first contact this server to retrieve metadata and then execute **Read** or **Write** actions. An immediate implication is that the administrator can be placed in this server, having the benefit of total control over the lists and the possibility to synchronizing accesses. However, this does not force all the traffic to flow through the server, as users can interact with both the server and the CSP at the same time. The main advantage of this solution is the possibility to personally regulate access to lists and tuples. Also, metadata would be hidden from the CSP and monetary costs would be lowered. Another benefit is that this solution would speed up AC policies maintenance operations and clients' queries, being the metadata locally stored and not fetched from a remote location.

  On the other hand, relying on an internal server introduces a SPOF and makes the architecture less robust. Also, it would add complexity to the deployment. In fact, the company would have to configure and manage the server. Another point to consider is that the server would receive many requests from the clients fetching tuples and lists. Depending on the capabilities of the

server, it could become a bottleneck. This introduces other issues concerning the scalability of the architecture and further complicates its deployment.

- **Cloud**: if metadata is stored in the Cloud, they would always be available and online, dodging the SPOF scenario and enhancing robustness. Moreover, the company would avoid the setup and management of a dedicated server, making the entire architecture easier to adopt and deploy. Lists and tuples could be stored in a DB or as metadata of files. The last option would get rid of a DB with the benefit of storing all the information in just one site. For instance, to read a file, a user could just download it from the Cloud. All necessary information to decrypt it would already be present as metadata of the file itself.

  Unfortunately, this may not be possible because of space limitations (e.g., AWS limits 2KB of metadata for each file at most [39]). Furthermore, role tuples would be redundantly replicated across the files creating synchronization issues. Moreover, because of the IBE-IBS algorithm or other choices of the company, the metadata may contain sensitive information that the company may not be willing to store in the Cloud. This limits only to lists because tuples do not contain sensitive data. Furthermore, the more the CSP is used for storage and services, the more the monetary costs. Moreover, this has an impact on efficiency as well: the farther the data, the higher the time requested for modifying the cryptographic AC policy or executing **Read** or **Write** operations.

- **Hybrid Solution**: since lists and tuples are related but storage-independent, we can save them in two different locations. For instance, lists can be stored in the Cloud while tuples can be stored in a local server within the company, thus closer to the clients. Since tuples and intensively queried when reading and modifying the policy, this would speed up both operations but not as if the lists were locally stored too. Lists, less frequently accessed, can be stored in the Cloud. The only issue is about the privacy of metadata, like the lists of users' and roles' IDs.

  It is possible also to store lists in the local server and tuples in the Cloud. As it is possible to imagine, this configuration has opposite pros and cons with respect to the one illustrated above.

We do not analyze a further hybrid scenario in which we split the lists and tuples to have part of them in a server within the company and part in the Cloud. In fact, this solution just sharpens the drawbacks just discussed and introduces multiple interactions and synchronization issues.

In Table 5.1 (page 45) there is an aggregated view of the pros and cons with respect to the storage location of all lists and tuples. Note that the Keys Location, Control on Accesses and Multi-Device features are not influenced by this degree of freedom.

5.3.2 Clients' access to the CSP

Regardless of where metadata are stored, clients have different ways to access files in the CSP. As is it possible to image, this has a significant impact on the architecture, especially the relationships among entities. For instance, it may lead to the creation of a proxy to mediate all interactions, or it could require to extend the scheme to distribute temporary credentials to the clients. There are two options for clients' access to the CSP:

- **Dedicated Proxy**: the company can configure a trusted proxy within its boundary. Then, clients would ask it to handle requests on their behalf. This proxy would take the role of the administrator and it would have total control over accesses to files. For instance, it could easily lock

Feature	Dedicated Server	Cloud	Hybrid Solution	
			Lists on Cloud	Tuples on Cloud
Deployment Readiness	-	+	--	--
Scalability	--	++	-	+
Workload Management	--	++	-	+
Robustness	-	+	-	-
Control on Metadata	++	--	-	+
Policy Maintanance and Queries Speed	++	--	+	-
Monetary Savings	++	--	-	-

Table 5.1: Pros and Cons for Metadata Storage Location

two writers to synchronize their actions. However, in this scenario, there are no other benefits. Instead, the drawbacks would be a noticeable overhead for bandwidth because of the unique point of upload-download of files and the creation of a SPOF. An independent consideration is that the CryptoAC entity could be placed either in this proxy or in another program installed in the clients. The former option may create another bottleneck, this time because of the burden of the encryption and decryption operations. This would certainly be an efficiency issue, even though it would allow using also light devices (e.g., smartphones). Moreover, all the private keys of all users would be stored together in the same machine. Even though feasible, this is a delicate and risky scenario to handle. The latter option, instead, requires the company to set up both the proxy and the software on employees' computers. This may not be worth the only advantage of synchronizing accesses to files.

- **Direct Connection to the CSP**: clients can directly access the CSP without the setup of a proxy in the middle. Each client would install the software needed to connect to the CSP and directly fetch and upload files. This configuration of the architecture releases control over the clients' actions and gains flexibility and simplicity. Removing the proxy, we would also increase the speed of the scheme with respect to the clients since we remove an extra hop in the architecture. An independent choice is the kind of credentials clients use to authenticate to the CSP. In fact, there can be temporary or personal credentials. In the first case, the administrator, as a server, is responsible for distributing temporary credentials for accessing the CSP. These should be created with defined privileges so to allow a client to access only the files he/she requests, making it easier to handle permissions and policies and enhancing AC. The disadvantage is that it would not be possible to exploit traditional AC mechanisms offered by the CSP through the identification of the client with personal credentials. Instead, policies should be parsed and enforced directly by the administrator and this is an error-prone approach. Again, this scenario introduces a SPOF since, without the server, clients cannot receive credentials to authenticate to the CSP. Moreover, it requires the development and maintenance of two different programs: one for the server to generate credentials and one for running CryptoAC in the clients. Otherwise, as supported by assumption CA4, clients can have personal credentials. This may have a limit for what concerns the number of registered users, but it would possible to enforce AC both through

the service offered by the CSP and through cryptography as well.

In Table 5.2 (page 46) there is an aggregated view of the pros and cons with respect to the way clients access to the CSP. Note that the Control on Metadata, Monetary Savings and Policy Maintenance and Queries Speed features are not influenced by this degree of freedom.

Feature	Dedicated Proxy	Direct Connection	
		Personal Credentials	Temporary Credentials
Deployment Readiness	-	+	--
Keys Location	-	+	+
Scalability	--	++	-
Workload Management	--	++	-
Robustness	-	+	-
Multi-Device	+	-	-
Control on Accesses	++	--	+

Table 5.2: Pros and Cons for Clients' Access to the CSP

5.4 Two Architectures

In the previous discussion, we showed that the more the administrator is involved (e.g., by storing metadata), the higher the possibility to introduce a SPOF. Nevertheless, this reduces querying time, amount of trust given to the CSP and monetary expenses as well. Also, this increases the overall control enforceable by the administrator on the clients' actions. On the other hand, the more metadata are stored in the Cloud, the more expensive it becomes. However, this configuration eliminates the need for a proxy mediating the accesses between clients and the CSP. This is actually of great importance, not only for the difficulty of setting up the server. In fact, having a trusted proxy in the middle could make it unnecessary to have cryptographic AC at all: policies could be directly enforced by the proxy itself. Then, files could be encrypted with a single symmetric key, eliminating the overhead that the cryptographic AC scheme implies. However, as already discussed in [15], this approach increases the complexity of the deployment by inserting another entity in the communication flow (i.e. the proxy itself). Depending on the company adopting such an architecture, there can be different issues: small companies may have difficulties in the configuration of the server, making the whole solution less appealing. On the other hand, big companies could have problems in the synchronization of policies across several Proxies. Keeping in mind all of these reflections, we analyzed the twelve different architectures that come from the available options for the two degrees of freedom. We decided to further discuss two specific architectures. We configure them so that they emphasize opposite features from each other. The first one we present centralizes controls and accesses as much as possible, while the second enhances flexibility and scalability.

5.4.1 Proxy Architecture

In the first architecture, every client connects to a single proxy, placed inside the boundary of the company and thus deemed secure and trusted. Clients retrieve lists and tuples needed for decrypting

files directly from this proxy. This architecture enhances control over efficiency, thus all communications are mediated by the proxy. Also, all data flow through it and metadata are stored there as well. The company could also decide to store lists and tuples in the CSP; this is possible since the design expects the data to be stored in a DB. Thus, it can be hosted either by the CSP or within the boundary of the company as another micro-service. Note that it is not necessary to transmit files through the proxy since this communication can also happen directly. In this case, remember that clients should be given either personal or temporary credentials.

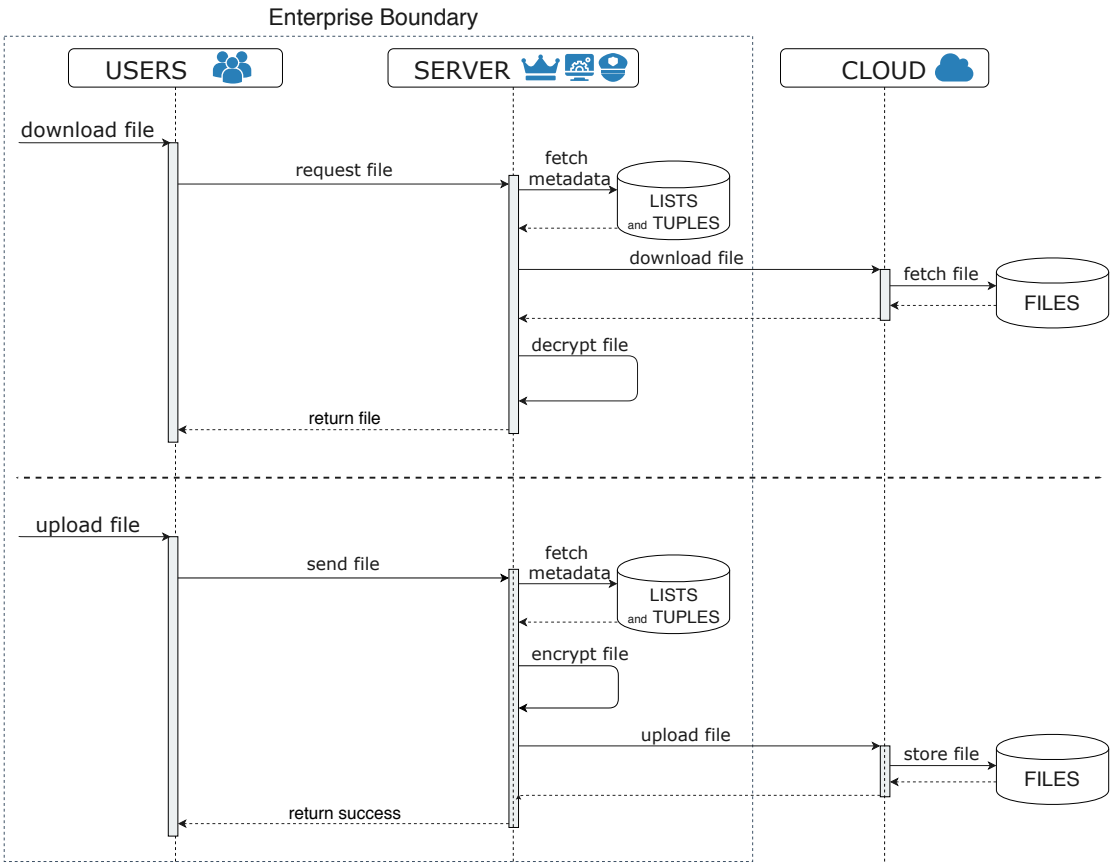


Figure 5.1: Storage of Lists and Tuples in Remote Solution

In Figure 5.1 (page 47) we report the UML sequence diagram related to this architecture for the two most important operations, **Read** and **Write**. The functioning is pretty straightforward: the client requests a file to the proxy. This parses the tuples encoding the policy and, if authorized, it downloads the file from the Cloud and then decrypts it. Eventually, the plain-text is sent back to the client. The scenario for uploading a file is much similar. The only difference is the presence of the RM that, as you can see in Figure 5.1, is represented by the proxy. As you can notice, this architecture greatly emphasizes control. In fact, it is possible to enforce both cryptographic and traditional AC, the former through the cryptographic scheme and the latter through traditional AC schema in the proxy. Besides, the proxy can handle synchronization of write operations and lock resources. The disadvantages were amply described before: lack of flexibility, complexity in the deployment of a dedicated proxy and the creation of a bottleneck and a SPOF are all drawbacks to accept when choosing this arrangement.

5.4.2 Straight Architecture

The second architecture is instead simpler and with direct interactions toward the CSP. In this architecture, the administrator and the clients connect to the CSP with personal credentials without passing through an intermediary proxy. The CSP holds all the files and also metadata. This architecture emphasizes flexibility and readiness in the deployment since no further configuration is required except for the installation of CryptoAC on clients' machines. Also, the whole architecture becomes perfectly scalable, since the company does not have to manage any server or directly handle any employee's request. The RM entity is placed in the Cloud as supported by CA1 and CA2. However, since everything is moved to the Cloud, this solution is on contrary more expensive in terms of money than the previous one. Again, since metadata are designed to be in a DB, it would be possible to store them in another micro-service within the boundary of the company. Moreover, the administrator could also distribute temporary credentials. However, this comes at the cost of setting up a server and creating a SPOF.

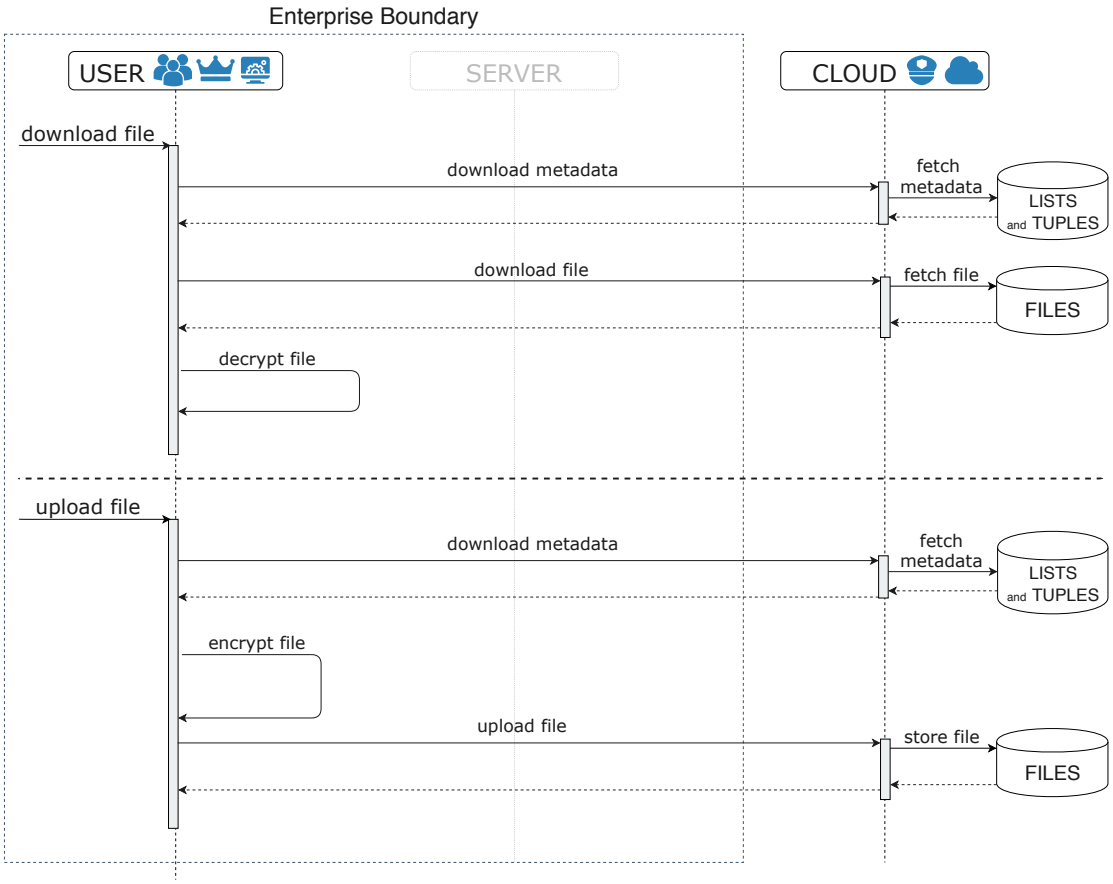


Figure 5.2: Storage of Lists and Tuples in Local Solution

In Figure 5.2 (page 48) we report the UML sequence diagram for the **Read** and **Write** operations. However, this time the server is faded out because it does not actually exist: every action occurs through a direct connection. Clients are authenticated by the CSP, that is also responsible for making data available directly to them. The RM entity is then moved to the CSP, and checks are there performed and enforced.

5.4.3 Comparison of Proxy and Straight Architectures

In this section, we summarize the pros and cons of the proposed architectures. Note that these are opposite in nature, being the remote centralized with the proxy and the second one decentralized. However, they can be adapted to create a hybrid solution to enhance different features of the architecture. As amply described above, each degree of freedom is independent of the other and pros and cons can be combined to better fit the needs of a specific use case. Table 5.3 (page 49) reports pros and cons with respect to the features for both architectures. This table was composed by summing up **+** and **-** signs from the two tables 5.1 and 5.2 presented above.

Feature	Proxy Architecture	Straight Architecture
Deployment Readiness	--	++
Keys Location	-	+
Scalability	----	++++
Workload Management	----	++++
Robustness	--	++
Control on Metadata	++	--
Policy Maintanance and Queries Speed	++	--
Monetary Savings	++	--
Multi-Device	+	-
Control on Accesses	++	--

Table 5.3: Pros and Cons for the Two Proposed Architectures

Chapter 6

Implementation

In Chapter 5 we discussed several architectures for the deployment of the cryptographic $RBAC_0$ scheme presented in Chapter 3. In order to give concreteness to our discussion, we now illustrate a beta implementation of the Straight architecture. We chose this because simpler to develop and test with respect to the others.

In this Chapter, we start by showing how we combined the five entities of the cryptographic $RBAC_0$ scheme with actual software. Then we discuss some peculiar aspects of the implementation and the interfacing with a specific CSP. We choose to interact with AWS, one of the most widely used CSPs. Eventually, we prove the validity of our implementation by demonstrating how it respects all the requirements formulated in Chapter 4.

6.1 Entities and Software

In this section, we describe how the five abstract entities match the software we decided to use for the concrete implementation. In Figure 6.1 (page 51) we report the three software components we exploit in our implementation.

(a) Browser (b) AWS (c) Docker

Figure 6.1: Cryptographic AC Related Software Components

In Figure 6.2 (page 52) we provide a graphical view of how entities and software are combined. CryptoAC runs within a Docker container. Note that, since there is no proxy, each client has to install Docker on his/her computer. An important point worth mentioning is that the software is the same for all clients and administrator as well. This is possible since we rely on AC policies offered by the CSP to allow or deny administrative operations, like the deletion of a file. Both the clients and the administrator access CryptoAC through the browser like as connecting to any website. The CSP is represented by AWS since this is our target for the concrete implementation. AWS contains both the CSP and the RM. The first one stores files, while the second is invoked whenever a client upload a file.

To be as thorough as possible, we translated step-by-step the pseudocode provided in [15] into

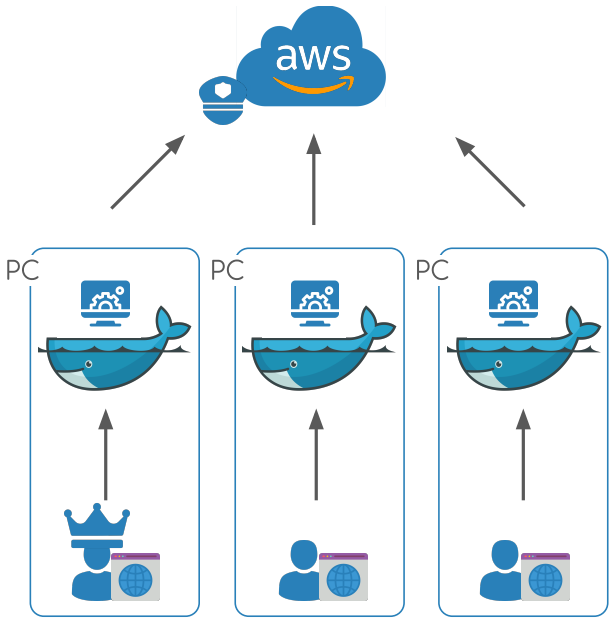


Figure 6.2: Straight Architecture - Entities and Software

the concrete implementation, stating which entity performs which operation. In this way, we were able to assign each operation to an entity and therefore to the software in charge of executing it. In Figure 6.3 (page 53) we provide an excerpt of such a document. The black rows correspond to the original pseudocode, the blue ones to our interpretation of the algorithm from the point of view of our architectures.

The clients and the administrator are impersonated by real employees of the company. The CSP is AWS. Therefore, the only two programs we had to develop are related to the other two entities, CryptoAC and the RM.

6.2 CryptoAC

CryptoAC is written in Java and packaged in a Docker image that can run in any OS. The User Interface (UI) is rendered through web technologies (HTML, CSS and Javascript) so to not be bounded to any specific platform. Before using CryptoAC, the administrator has to contact the CSP and configure the needed services. In this case, there is the need for a service through which download and upload files and a DB service. After this phase, as assumed in CA4, each client is given personal credentials to access the services of the configured CSP.

The first time a client access to CryptoAC, he/she has to authenticate and configure parameters like his/her personal credentials for the connection to the CSP, the URL of the DB and the service for download and upload of files. Clients' keypairs are by default generated inside CryptoAC. The private key is stored encrypted on the filesystem with a password chosen by the client itself and it is never transmitted across the network. CryptoAC saves all these data internally under the ID of the client, creating a personal profile. The same procedure is followed by the administrator.

After the creation of the profile, the client can immediately start interacting with the CSP by adding, reading or writing files, according to the cryptographic RBAC_0 policy. The administrator can add or delete users and roles and distribute or revoke permissions. All these functionalities can be accessed either by the browser interface (see Figure 6.4 page 54 and Appendix B) or through REST

: $delU(u)$ - Delete User
* Delete $\mathbf{k}_u^{\mathbf{enc}}$ and $\mathbf{k}_u^{\mathbf{ver}}$
  Step 1 (administrator): the administrator deletes the entry with Username u from the table **users** in the DB
* For every role r that u is a member of:
  Step 2 (administrator): the administrator fetches the role tuples with Username u from the table **roleTuples** in the DB, and for each matching entry:
  - $revokeU(u, r)$
    Step 3 (administrator): the administrator invokes the $revokeU(u, r)$ function

: $addP_u(fn, f)$ - User adds a File
* Generate symmetric key $k \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$
  Step 1 (client): User generates a symmetric key $k \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$
* Send $\langle \mathsf{F}, fn, 1, \mathbf{Enc}_k^{\mathbf{Sym}}(f), \mathbf{Sign}_{\mathbf{k}_u^{\mathbf{sig}}}^{\mathbf{Sig}} \rangle$ and $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, \mathbf{Enc}_{SU}^{\mathbf{Pub}}(k), u, \mathbf{Sign}_{\mathbf{k}_u^{\mathbf{sig}}}^{\mathbf{Sig}} \rangle$ to R.M.
  Step 2 (client): User creates and sign the tuples. Then he/she uploads file and metadata to the RM
* The R.M. receives $\langle \mathsf{F}, fn, 1, c, sig \rangle$ and $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, c', u, sig' \rangle$ and verifies that the tuples are well-formed and the signatures are valid, i.e. , $\mathbf{Ver}_u^{\mathbf{Sig}}(\langle \mathsf{F}, fn, 1, c \rangle, sig) = 1$ and $\mathbf{Ver}_u^{\mathbf{Sig}}(\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, c', u \rangle, sig') = 1$.
  Step 3 (RM): The Reference Monitor checks the integrity of the data as expected in the algorithm
* If verification is successful, the R.M. adds $(fn, 1)$ to FILES and stores $\langle \mathsf{F}, fn, 1, c \rangle$ and $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, c', u, sig' \rangle$
  Step 4 (RM): If the outcome is successfull, the RM adds $(fn, 1)$ to **files** table in the DB
  Step 5 (CSP): If the outcome is successfull, the CSP stores the file

Figure 6.3: Operation-wise Translation of Pseudocode

APIs.

Other interesting aspects to describe are related to the configurability of CryptoAC, the interfacing with CSPs and the storage of the metadata encoding the policy.

6.2.1 Multi-Architecture

As described in Section 6.1, each client runs Docker with CryptoAC inside. The access to its functionalities occurs through the browser interface where clients authenticate and configure their own profile. Therefore, it is possible to think of CryptoAC as a website. Suppose now that CryptoAC is not installed in clients' computers but instead in an internal server configured by the company. Every client would access CryptoAC as before using the very same browser, but the architecture would be centralized. Therefore, we can swiftly and effortlessly change architecture by introducing a proxy while keeping the very same software.

As shown Figure 6.5 (page 55), it is also possible to have a hybrid architecture with some instances of CryptoAC running on clients' computers and one deployed in an internal server, so to allow even smartphones or temporary users to join the architecture.

6.2.2 Data Access Object Pattern

To be able to support multiple CSPs, we employed the Data Access Object (DAO) pattern. In general, the DAO pattern consists of an abstraction of the functions needed to interact with the underlying storage solution without exposing low-level details. This allowed us to decouple the interaction with

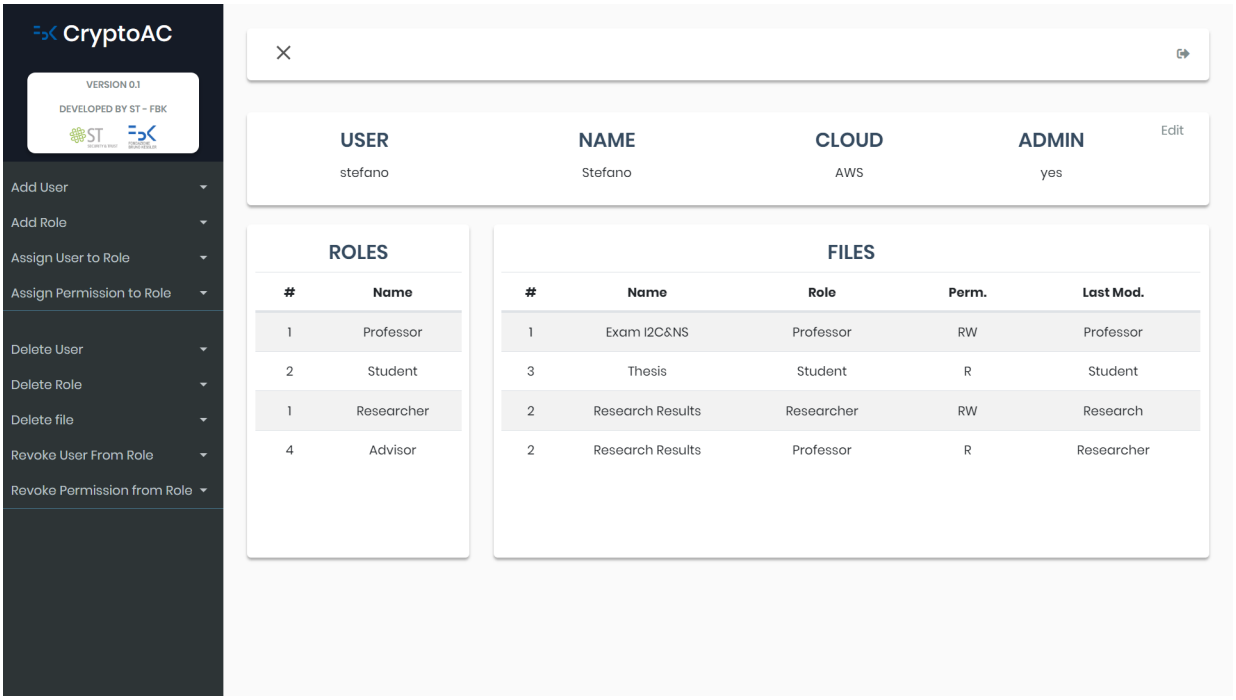


Figure 6.4: CryptoAC Interface with Administrator's Operations on the Left

metadata and files from the sheer logic of the algorithm. The result is a Java interface composed by a list of functions to implement to interface with the CSP, along with complete documentation. To add support to a new CSP, one just needs to create a new subclass implementing those functions. Figure 6.6 (page 56) shows the UML sequence diagram for the `Add` file action. The user interacts with the **DAOCloud** interface to retrieve metadata and upload the file. Being an interface, this sequence of operations in independent from the underlying implementation.

We have already implemented the functions interacting with metadata in case these are stored in a MySQL DB. Therefore, the effort required to interface with a new CSP is reduced to the only implementation of the upload and download functions specific to the new CSP.

6.2.3 Metadata Hiding

In a company, it may be important that employees do not know what roles, files and permissions are present beside their own. Unfortunately, in [15] metadata are said to be publicly accessible and no mechanism is provided for hiding them. As we introduced the opposite requirement (CR7 - Metadata Hiding), we followed two procedures to hide metadata not related to the involved user: Views and Tokenization.

* **Views**: a view is the result set of a query over one or more tables. In our case, this mechanism is exploited to show to the users only their own assignments. In fact, users are granted permission to access the view but not the original table. In Figure 6.7 (page 57) we reported a snippet of SQL code for the creation of the view for the *roleTuples* table.

  The *user_specific_roleTuples* view (line 2) is composed by filtering the selected *roleTuples* table (lines 10-11) with the ID of the logged user (lines 12-13). Such an ID must be the same to the one of the corresponding entity in the $RBAC_0$ policy. Therefore, each user can access only the role tuples that contain his/her own ID. It is possible to apply this reasonsing also to the *permissionTuples* table.

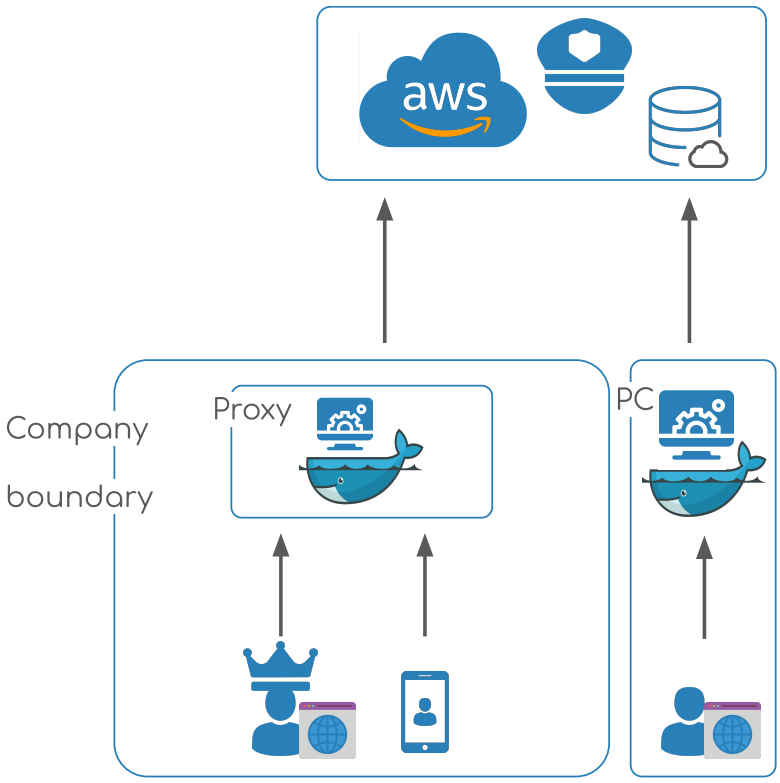


Figure 6.5: Hibrid Architecture with Proxy and Direct Connection

* **Tokenization**: we followed a process of Tokenization on the lists of users, roles and files. In general, a Tokenization process consists in the use of a meaningless piece of information (e.g., a random string) that acts as a reference pointing to the actual sensitive data (e.g., the ID of the entity). In our case, this is done by adding a random token in a new column in the tables of Users, Roles and Files, next to the ID of the element. Figure 6.8 (page 58) reports the code used for the creation of these tables and shows the three fields related to such tokens (lines 3-11-20).

  Remember that users have to search in these tables for public keys of entities to validate the digital signatures of tuples. Therefore, in the tuples, we replaced the ID of the signer with his/her token. In the DB, users are then denied access to the column holding such ID and are allowed to query data only through the token. In this way, no sensitive data is disclosed to the users, while the administrator has still the possibility to surf the complete version of the data.

6.3 Reference Monitor

The RM is a simpler software component with respect to CryptoAC. It consists of a single Java class that performs all the expected checks on the clients' operations. The RM has to be configured with several parameters, like the URL pointing to the DB of metadata and the credentials to access it. All these variables are stored in a configuration file that has to be modified prior to the deployment of the RM in the CSP. In fact, this information cannot be sent by the clients themselves, as it may be artificially crafted to bypass the integrity checks. For instance, a client could recreate a copy of the DB with bogus permissions and send the URL of this DB to the RM, that would validate the client's operation based on a fake RBAC_0 policy.

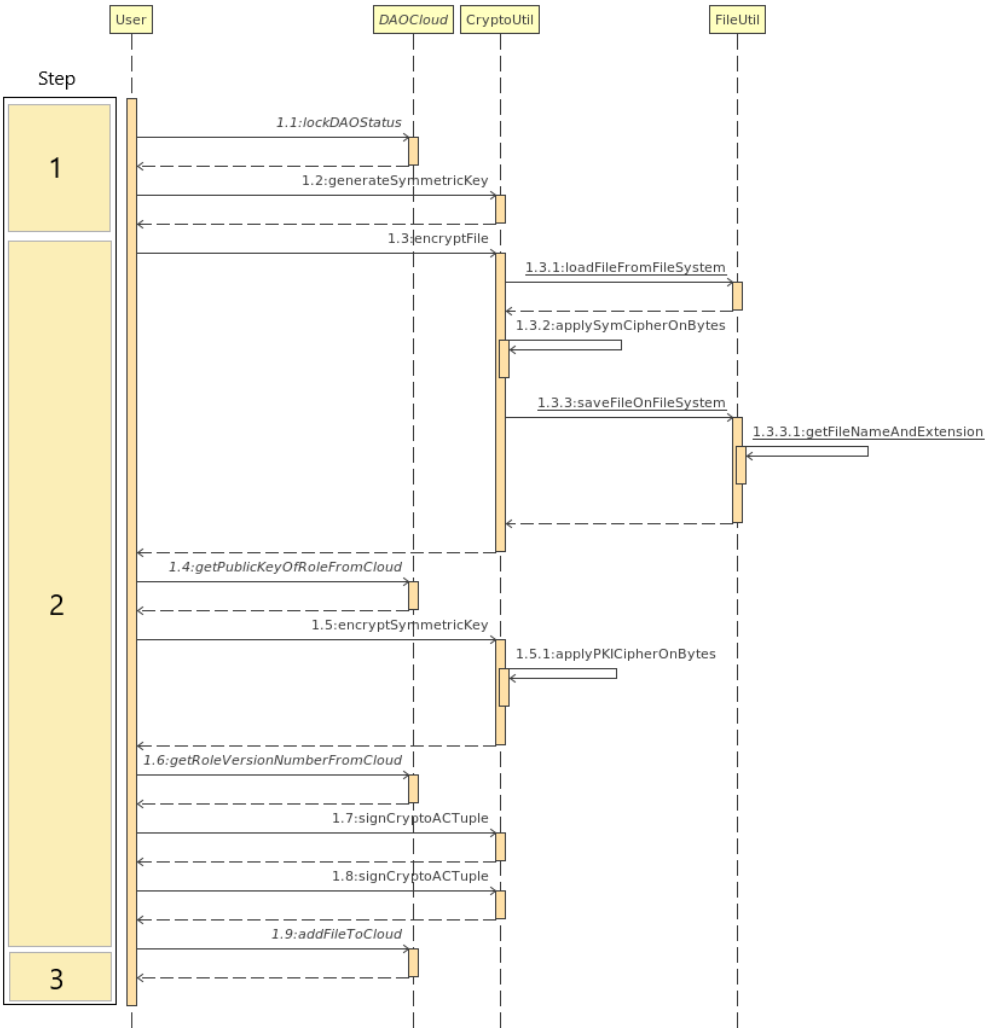


Figure 6.6: UML Sequence Diagram for the AddFile Action

6.4 Implementation with AWS

We implemented the **DAOCloud** interface to interact with AWS. We used S3 [40] as service for the storage of files inside folders that AWS named "buckets". Then we exploited RDS [41] as DB service for the storing of metadata and the Lambda Functions [36] service for the RM. We configured IAM [9] policies to limit clients' permissions so to allow them to interact only with stored files.

6.4.1 Lambda Function

In the proposed architecture, remember that the `Read` action occurs directly between clients and CSP. About the `Write` action, clients directly invoke the RM (i.e. the Lambda function) by submitting their request along with the new file. However, since Lamba functions are billed based on execution time, this solution may quickly become expensive due to the slow upload of large files. As reported in Figure 6.9 (page 59), we created two distinct buckets, one for uploading and the other for downloading of files. In this way, clients first upload their file in the temporary bucket. Next, they invoke the Lambda function that moves the file in the definitive bucket, from which it will be accessible by all other clients.

```
01 |     /* users can access only their role tuples */
02 |     CREATE VIEW `user_specific_roleTuples` (`username`, `roleName`, `
       roleVersionNumber`, `encryptedRolePublicKey`, `encryptedRolePrivateKey`, `
       signature`) AS
03 |     SELECT
04 |     `roleTuples`.`username` AS `username`,
05 |     `roleTuples`.`roleName` AS `roleName`,
06 |     `roleTuples`.`roleVersionNumber` AS `roleVersionNumber`,
07 |     `roleTuples`.`encryptedRolePublicKey` AS `encryptedRolePublicKey`,
08 |     `roleTuples`.`encryptedRolePrivateKey` AS `encryptedRolePrivateKey`,
09 |     `roleTuples`.`signature` AS `signature`
10 |     FROM
11 |     `roleTuples`
12 |     WHERE
13 |     `roleTuples`.`username` = (CONVERT( SUBSTRING_INDEX(USER(), '@', 1) USING
       UTF8MB4));
14 |
```

Figure 6.7: View of Table RoleTuples user-specific

6.4.2 IAM Policy

Since there are two buckets storing files, we need to configure the proper permissions through the IAM service. Figure 6.10 (page 60) reports the policy used for such a purpose. Every client can upload files in the first bucket (**cryptoac-files-temporary**, line 20) from which no one can download. After a positive check by the RM, the files are moved in the definitive bucket (**cryptoac-files**, line 12). Clients can only download from this bucket. These permissions are expressed by the **PutObject** (line 19) and the **GetObject** (line 8) keywords. The permission for invoking the Lambda is the **InvokeFunction** (line 9 and 13).

6.4.3 DB Scheme

We wrapped the file tuple as metadata of the file itself to keep them together in S3. This is possible since AWS allows for storing up to 2KB metadata with each file, plenty of space for a file tuple. In this way, users can download the file and immediately obtain further information like its version number or the token of the entity that lastly modified the file, with no further queries to the DB. This implies the removal of the table **fileTuples** from the DB.

6.5 Requirements Compliance

Even though the two degrees of freedom presented in Chapter 5 do not violate the requirements formulated in Chapter 4, it is still not proved that any derived implementation actually respects them. Therefore, in this Section, we discuss how our implementation respects each one of these requirements.

- **CR1 - Multi-CSP Support**: the implementation has to be able to interface with different CSPs. In order to respect this requirement, it should not rely on any peculiar service offered uniquely by a specific CSP. In fact, our architecture exploits only services covered by assumptions CA1 and CA2. In the AWS case, it uses S3 for the storage of files, RDS for the metadata and the Lambda functions for the RM. Concretely, multi-CSP support is provided with the DAO pattern.
- **CR2 - Multi-Configuration**: the possibility to have multiple architectures is already discussed

```
CREATE TABLE `users` (
`username` varchar(50),
`userToken` char(50) NOT NULL UNIQUE,
`publicKey` varchar(50) NOT NULL,
`isDeletedFlag` tinyint(4) NOT NULL,
PRIMARY KEY (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `roles` (
`roleName` varchar(50),
`roleToken` char(50) NOT NULL UNIQUE,
`publicKey` varchar(50) NOT NULL,
`roleVersionNumber` int(11) NOT NULL,
`isDeletedFlag` tinyint(4) NOT NULL,
PRIMARY KEY (`roleName`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `files` (
`fileName` varchar(50),
`fileToken` char(50) NOT NULL UNIQUE,
`encryptFileVersionNumber` int(11) NOT NULL,
PRIMARY KEY (`fileName`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 6.8: Creation of Lists of Users, Roles and Files

and demonstrated at high-level in Chapter 5 when presenting the possible architectures. Moreover, we saw in Section 6.2 that CryptoAC itself can be configured to support multiple architectures.

- **CR3 - Micro-Services**: the use of Docker allowed us to develop a modular implementation: CryptoAC, RM and DB can run as micro-services. Concretely, CryptoAC exposes REST APIs. We implemented more than 20 different APIs offering different functionalities, from managing the cryptographic AC policy to uploading and downloading files.

- **CR4 - Multi-Platform**: while the RM runs as Lambda function in AWS, CryptoAC is written in Java and packaged in a Docker image that can run in any OS. The UI is rendered through web technologies so to not be bounded to any specific platform.

- **CR5 - Accountability**: this requirement is partially fulfilled by the mechanism of digital signatures inserted in the cryptographic AC scheme. In fact, almost every action produces tuples that are then digitally signed by the entity that performed that operation. This could be increased also by not eliminating old tuples like expected by the scheme. Instead, these could be moved in another table in the database or just flagged. This would allow creating a history that could serve as an audit log. Also, it would be interesting to make users and not roles digitally sign the new files after a `Write` action. In any case, all major CSPs provide a complete logging service that we exploit by considering personal credentials for clients [42, 43].

- **CR6 - Secure Keys Handling**: Clients' keypairs are by default generated inside CryptoAC. After use, the private key is stored encrypted on the filesystem. The encryption key is a password chosen by the client itself. Considering then private keys are never transmitted across the network, we can conclude that they are securely handled.

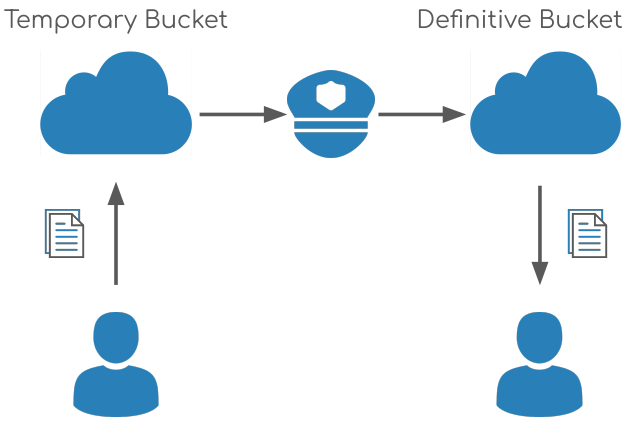


Figure 6.9: Upload of Files to Lambda with Two Buckets

- **CR7 - Metadata Hiding**: the use of Views and Tokenization described in Section 6.2 allowed us to hide the policy to the clients revealing only their own assignments.
- **CR8 - CSP Cannot Read Files**: to prevent the CSP from reading the content of files the company uploads, our implementation blocks both passive and active access. In fact, files are always encrypted client-side before being uploaded to the Cloud. Moreover, no private cryptographic key is sent or stored in the Cloud. The RM just checks the integrity of digital signatures of the tuples with public keys, so also this entity cannot access any encrypted data.
- **CR9 - Policy Integrity**: the users could try to tamper with the policy (i.e. the tuples) to obtain `Write` permission over a file. As explained in Section 4.3.2, it is not possible to gain `Read` permission since this action is enforced cryptographically. In order to prevent users from modifying the policy or trying to insert fake tuples, digital signatures are checked both when tuples are stored in and when they are retrieved from the DB. Any new tuple create by users is checked by the RM that runs in the CSP. Remember that the CSP is assumed to be trusted for such tasks by assumption CA2. We can conclude that any tampering on the policy can be easily prevented.
- **CR10 - Secure Communication Channels**: all the communications occur through encrypted channels, exploiting cryptographic protocols for securing information. By changing architecture, the number of channels may vary. Still, each communication is secured through means of encryption by using standard technologies developed for such purpose (e.g., TLS).
- **CR11: No Spurious Downloads from Users**: as the authors suggested [15], we used unguessable file names randomly produced. These are the tokens derived from the process of Tokenization described in Section 6.2.3. In order to download a file, a user has to know its token. The only way to know the token of a file is to have a valid permission over the file itself. Therefore, users can not download files they are not assigned to.
- **CR12: Resolving Concurrency**: there are three cases in which there might be a concurrency issue: when (i) two users write on a file at the same time or (ii) a user starts a `Write` actions at the moment the permission is revoked by the admin or (iii) the admin executes two revoke operations simultaneously. In the first case, the only problem is that a user overwrites the changes

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "VisualEditor0",
"Effect": "Allow",
"Action": [
"s3:GetObject",
"lambda:InvokeFunction"
],
"Resource": [
"arn:aws:s3:::cryptoac-files/*",
"arn:aws:lambda:eu-central-1:503968272143:function:
CryptographicRBACLambda"
]
},
{
"Sid": "VisualEditor1",
"Effect": "Allow",
"Action": "s3:PutObject",
"Resource": "arn:aws:s3:::cryptoac-files-temporary/*"
}
]
}

```

Figure 6.10: IAM Policy in AWS

of the previous one, but the operation is atomic by itself. Therefore, there is no concurrency problem. In the second case, the user operates with the files and the admin with the tuples. Being modifying different data, we can conclude that there is no concurrency problem. To handle the third case, we rely on the fact that the administrator is a single entity within the company. Therefore, he/she cannot perform two operations at the same time. Also, we developed a mechanism inside CryptoAC that prevents the administrator to insert another operation until the last one is completed.

Chapter 7

Conclusions and Future Work

In this thesis, we presented the first attempt to fill the gap between the many abstract cryptographic AC schemes available in the literature and a pragmatic approach for a real-world deployment. As illustrated in Chapter 1, Cloud Computing is nowadays a major innovation driving processes and strategies of many companies. According to Eurostat, storage of files is the second most used service in the Cloud [5]. Given the several incidents already occured [10, 11, 44], it is crucial to preserve the confidentiality of files stored in the Cloud while maintaining the possibility to let employees share resources and operate together. Cryptographic AC is the natural solution to this situation.

Unfortunately, the literature offers only abstract approaches with little focus on their actual practicality. We shift the focus from proposing yet another cryptographic AC scheme to study in deep how to implement one. We uncovered several assumptions and requirements for our architecture. Even though derived from our use case, they could all serve as guidelines for other future implementations. Furthermore, we illustrated the possible ways in which entities can relate to each other. We specified what are the options that influence the architecture through two degrees of freedom. We analyzed the advantages and drawbacks of twelve different architectures but did not constrain any choice to an eventual company employing our solution. In fact, the architecture is highly customizable to cover as many use cases as possible and fit different needs that different companies may have.

To give concreteness to our discussion, we developed a tool named CryptoAC. As a starting point, we interfaced CryptoAC with AWS as CSP. Since we wanted to embed as much flexibility as possible, we did not limit the company to the use of a specific CSP or architecture. In fact, CryptoAC can easily interface multiple CSPs by just implementing two functions and can support multiple architectures with a simple, if not completely null, configuration effort. The GUI is available through a browser interface and the functionalities are accessible through RESTful APIs. We presented a detailed view of the issues and challenges to take into account when actually implementing a cryptographic AC scheme and we hope that CryptoAC will actually be used for enforcing cryptographic AC policies in the Cloud.

7.1 Future Work

Minor Improvements It would be interesting to perform a concrete analysis of the performance of CryptoAC to have a comparison with the one simulated in [15]. Moreover, it could be useful to add support to other CSPs and refine the already present documentation on how to do it. We are also planning to enhance the UI by including the possibility to import/export users' profiles and by furtherly detailing the dashboard of the administrator.

Schema Expressiveness We are working with the authors in [15] to implement other functional extensions to both their scheme and our implementation. In particular, we want to give users the possibility to directly access files without having to assume a role. Also, we aim at improving the expressivity of the enforceable policy by considering how to deny access to a specific user.

Hybrid Approach As seen in this thesis, cryptographic AC can be employed to substitute traditional AC policies. If the main priority is the confidentiality of data, cryptographic AC is a perfect solution to handle Honest but Curious CSPs. However, cryptography is effective but not efficient when applied to AC policies [15]. Therefore, a scheme relying on cryptographic AC only is likely to be unusable because of poor performance and efficiency issues. Since we are deeply concerned about the usability of our solution, as future work we pose the possibility to enforce a hybrid AC scheme. This means that the company can protect the most sensitive data through cryptography and leave the CSP to enforce AC policies over other files. In particular, we want to integrate CryptoAC with SecurePG [45]. This is a tool to assist administrators to specify high-level language AC policies. Then, it automatically verifies, translates and enforces these policies on a specific CSP. As of now, it is possible to write a single AC policy and automatically derive it for AWS and OpenStack. A successful integration of CryptoAC and SecurePG would allow enforcing a hybrid AC scheme, perhaps the only way to make cryptographic AC actually usable.

Bibliography

[1] Brian Hayes. *Cloud Computing*. In: "Commun. ACM" 51.7 (July 2008), pp. 9–11. ISSN: 0001-0782. DOI: 10.1145/1364782.1364786. URL: http://doi.acm.org/10.1145/1364782.1364786.

[2] Massachusetts Institute of Technology MIT. *Who Coined 'Cloud Computing'? Now that every technology company in America seems to be selling cloud computing, we decided to find out where it all began*. URL: https://www.technologyreview.com/s/425970/who-coined-cloud-computing/ (visited on June 27, 2019).

[3] Gartner. *Gartner Hype Cycle*. URL: https://www.gartner.com/en/research/methodologies/gartner-hype-cycle (visited on June 27, 2019).

[4] Peter M. Mell and Timothy Grance. *SP 800-145. The NIST Definition of Cloud Computing*. Gaithersburg, MD, United States, 2011.

[5] the statistical office of the European Union Eurostat. *Cloud computing services used by more than one out of four enterprises in the EU*. URL: https://ec.europa.eu/eurostat/documents/2995521/9447642/9-13122018-BP-EN.pdf/731844ac-86ad-4095-b188-e03f9f713235 (visited on June 27, 2019).

[6] Agenzia per l'Italia Digitale AGID. *Il Cloud della Pubblica Amministrazione. L'adozione del modello cloud computing nelle amministrazioni italiane*. URL: https://cloud.italia.it (visited on June 27, 2019).

[7] Amazon Web Services. *Shared Responsibility Model*. URL: https://aws.amazon.com/compliance/shared-responsibility-model/ (visited on July 4, 2019).

[8] Microsoft. *Manage access to Azure resources using RBAC and the Azure portal*. URL: https://docs.microsoft.com/en-gb/azure/role-based-access-control/role-assignments-portal (visited on June 13, 2019).

[9] Amazon Web Services. *AWS Identity and Access Management (IAM)*. URL: https://aws.amazon.com/iam/ (visited on June 13, 2019).

[10] Lyly Hay of Wired Newman. *The Scarily Common Screw-Up that Exposed 198 Million Voter Records*. URL: https://www.wired.com/story/voter-records-exposed-database/ (visited on June 27, 2019).

[11] Lyly Hay of Wired Newman. *Blame Human Error for WWE and Verizon's Massiva Data Exposures*. URL: https://www.wired.com/story/amazon-s3-data-exposure/ (visited on June 27, 2019).

[12] Breach Level Index. *Data Breach Statistics. Data Records Lost or Stolen since 2013*. URL: https://breachlevelindex.com/ (visited on June 27, 2019).

[13] Amazon Web Services. *Amazon S3 Default Encryption for S3 Buckets*. URL: https://docs.aws.amazon.com/AmazonS3/latest/dev/bucket-encryption.html (visited on June 27, 2019).

[14] Microsoft. *OneDrive Personal Vault brings added security to your most important files and OneDrive gets additional storage options*. URL: https://www.microsoft.com/en-us/microsoft-365/blog/2019/06/25/onedrive-personal-vault-added-security-onedrive-additional-storage/ (visited on June 27, 2019).

[15] William C. Garrison, Adam Shull, Steven Myers, and Adam J. Lee. *On the Practicality of Cryptographically Enforcing Dynamic Access Control Policies in the Cloud*. In: "Proceedings of 2016 IEEE Symposium on Security and Privacy (SP '16)". May 2016, pp. 819–838. DOI: 10.1109/SP.2016.54.

[16] Lan Zhou, Vijay Varadharajan, and Michael Hitchens. *Achieving Secure Role-Based Access Control on Encrypted Data in Cloud Storage*. In: "Information Forensics and Security, IEEE Transactions on" 8 (Dec. 2013), pp. 1947–1960. DOI: 10.1109/TIFS.2013.2286456.

[17] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. *Attribute-based encryption for fine-grained access control of encrypted data*. In: "Proceedings of the ACM Conference on Computer and Communications Security". Jan. 2006, pp. 89–98. DOI: 10.1145/1180405.1180418.

[18] Sascha Muller and Stefan Katzenbeisser. *Hiding the Policy in Cryptographic Access Control*. In: "Security and Trust Management". 2012, pp. 90–105. DOI: 10.1007/978-3-642-29963-6_8.

[19] Mikhail J. Atallah, Marina Blanton, Nelly Fazio, and Keith B. Frikken. *Dynamic and Efficient Key Management for Access Hierarchies*. In: "ACM Trans. Inf. Syst. Secur." 12.3 (Jan. 2009), 18:1–18:43. ISSN: 1094-9224. DOI: 10.1145/1455526.1455531. URL: http://doi.acm.org/10.1145/1455526.1455531.

[20] Somchart Fugkeaw and Hiroyuk Sato. *Design and Implementation of Collaborative Ciphertext-Policy Attribute-Role based Encryption for Data Access Control in Cloud*. In: 2015.

[21] Xin Jin, Ram Krishnan, and Ravi Sandhu. *A unified attribute-based access control model covering DAC, MAC and RBAC*. In: "Proceedings of the 26th Annual IFIP WG 11.3 conference on Data and Applications Security and Privacy". Vol. 7371. July 2012, pp. 41–55. DOI: 10.1007/978-3-642-31540-4_4.

[22] Ravi Sandhu. *Access control: principle and practice*. In: "Advances in Computers" 46 (Oct. 1998), pp. 237–286. DOI: 10.1016/S0065-2458(08)60206-5.

[23] Jeremy Horwitz and Ben Lynn. *Toward Hierarchical Identity-Based Encryption*. In: "Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques". Apr. 2002, pp. 466–481. DOI: 10.1007/3-540-46035-7_31.

[24] Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. *Encryption Policies for Regulating Access to Outsourced Data*. In: "ACM Transactions on Database Systems (TODS)" 35 (Apr. 2010), p. 12. DOI: 10.1145/1735886.1735891.

[25] Arnar Birgisson, Joe Gibbs Politz, Úlfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. *Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud*. In: Network and Distributed System Security Symposium. Jan. 2014. ISBN: 1-891562-35-5. DOI: 10.14722/ndss.2014.23212.

[26] John Bethencourt, Amit Sahai, and Brent Waters. *Ciphertext-Policy Attribute-Based Encryption*. In: "2007 IEEE Symposium on Security and Privacy (SP '07)". May 2007. DOI: 10.1109/SP.2007.11.

[27] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. *Bounded Ciphertext Policy Attribute Based Encryption*. In: "ICALP '08 Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part II". July 2008, pp. 579–591. DOI: 10.1007/978-3-540-70583-3_47.

[28] Rafail Ostrovsky, Amit Sahai, and Brent Waters. *Attribute-based encryption with non-monotonic access structures*. In: "CCS '07 Proceedings of the 14th ACM conference on Computer and communications security". Jan. 2007, pp. 195–203. DOI: 10.1145/1315245.1315270.

[29] Saman Zarandioon, Danfeng (Daphne) Yao, and Vinod Ganapathy. *K2C: Cryptographic Cloud Storage With Lazy Revocation and Anonymous Access*. In: "International Conference on Security and Privacy in Communication Systems". Vol. 96. Jan. 2012. DOI: 10.1007/978-3-642-31909-9_4.

[30] MongoDB. *MongoDB 4.2 Previewed At MongoDB World*. URL: https://mongodb.com/blog/post/mongodb-42-previewed-at-mongodb-world/ (visited on June 27, 2019).

[31] Yang Tang, Patrick P. C. Lee, John C. S. Lui, and Radia Perlman. *FADE: Secure Overlay Cloud Storage with File Assured Deletion*. In: "Security and Privacy in Communication Networks". 2010, pp. 380–397. DOI: 10.1007/978-3-642-16161-2_22.

[32] Valentin Ghita, Sergiu Costea, and Nicolae Tapus. *Implementation of Cryptographically Enforced RBAC*. In: "The Scientific Bulletin - University Politehnica of Bucharest" 79.2 (2017), pp. 9-3–102.

[33] Saiyu Qi and Yuanqing Zheng. *Crypt-DAC: Cryptographically Enforced Dynamic Access Control in the Cloud*. In: "IEEE Transactions on Dependable and Secure Computing" (2019). ISSN: 1545-5971. DOI: 10.1109/TDSC.2019.2908164.

[34] Julian Jang-Jaccard. *A Practical Client Application Based on Attribute Based Access Control for Untrusted Cloud Storage*. In: 11th International Conference on Security and its Applications. Jan. 2018, pp. 01–15. DOI: 10.5121/csit.2018.80101.

[35] Timothy L. Hinrichs, Diego Martinoia, W.C. Garrison, Adam J. Lee, Alessandro Panebianco, and Lenore Zuck. *Application-Sensitive Access Control Evaluation Using Parameterized Expressiveness*. In: June 2013, pp. 145–160. DOI: 10.1109/CSF.2013.17.

[36] Amazon Web Services. *AWS Lambda. Run Code Without Thinking About Servers. Pay Only for the Compute Time you Consume*. URL: https://aws.amazon.com/lambda/ (visited on June 13, 2019).

[37] Google. *Google Cloud Functions documentation*. URL: https://cloud.google.com/functions/docs/ (visited on June 13, 2019).

[38] IBM. *IBM Cloud Functions - Execute code on demand in a highly scalable serverless environment*. URL: https://www.ibm.com/cloud/functions (visited on June 13, 2019).

[39] Amazon Web Services. *AWS Service Limits*. URL: https://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html (visited on June 13, 2019).

[40] Amazon Web Services. *Amazon S3. Object storage built to store and retrieve any amount of data from anywhere*. URL: https://aws.amazon.com/s3 (visited on July 10, 2019).

[41] Amazon Web Services. *Amazon Relational Database Service (RDS). Set up, operate, and scale a relational database in the cloud with just a few clicks.* URL: https://aws.amazon.com/rds (visited on July 10, 2019).

[42] Amazon Web Services. *Centralized Logging*. URL: https://aws.amazon.com/it/solutions/centralized-logging/ (visited on July 9, 2019).

[43] Google. *Stackdriver Logging documentation*. URL: https://cloud.google.com/logging/docs/ (visited on July 9, 2019).

[44] Brian of Wired Barret. *Security News this Week: the Pentagon Left Data Exposed in the Cloud*. URL: https://www.wired.com/story/security-news-this-week-the-pentagon-left-data-exposed-in-the-cloud/ (visited on June 27, 2019).

[45] Umberto Morelli and Silvio Ranise. *Assisted Authoring, Analysis and Enforcement of Access Control Policies in the Cloud*. In: "IFIP SEC 2017 International Conference on ICT Systems Security and Privacy Protection". May 2017, pp. 296–309. ISBN: 978-3-319-58468-3. DOI: 10.1007/978-3-319-58469-0_20.

Acronyms

ABE Attribute Based Encryption. 18, 20, 21

AC Access Control. 1, 3, 7, 12–15, 17–21, 23–27, 29–33, 35, 37, 39, 40, 43, 45–48, 50, 53, 58, 63, 64

API Application Programming Interface. 3, 30, 32, 58, 59

AWS Amazon Web Services. 2, 3, 12–15, 20, 21, 39, 40, 46, 53, 54, 56, 58, 63, 64

CSP Cloud Service Provider. 11–15, 17–21, 30, 32, 34–41, 43–50, 53, 54, 56–58, 60, 61, 63, 64

DB Database. 2, 17, 53–61, 63

IAM Identity and Access Management. 2, 3, 12, 40, 54–56

IBE Identity Based Encryption. 26, 29, 37, 38, 41, 45, 46

IBS Identity Based Signature. 19, 26, 29, 37, 38, 41, 45, 46

OS Operative System. 17, 37, 38, 58

PKI Public Key Infrastructure. 26, 37, 38, 41

RBAC Role-Base Access Control. 19, 20

$RBAC_0$ Role-Base Access Control with no Hierarchy. 1, 12, 19, 23–26, 29, 30, 32, 33, 35, 38, 41, 59

RDS Relational Database Service. 54, 56

REST Representational State Transfer. 58

RM Reference Monitor. 18, 20, 24, 26, 28, 30, 32, 39–41, 44, 49, 50, 53–55, 57, 58, 61

SPOF Single Point Of Failure. 19, 46–48, 50

SQL Structured Query Language. 37, 39, 57, 59

TLS Transport Layer Security. 25, 61

UI user Interface. 64

UML Unified Modeling Language. 3, 49, 50, 57

URL Uniform Resource Locator. 53, 54

Appendix A Pseudocode from [15]

$addU(u)$
- User u generates encryption key pair $(\mathbf{k}_u^{\mathbf{enc}}, \mathbf{k}_u^{\mathbf{dec}}) \leftarrow \mathbf{Gen}^{\mathbf{Pub}}(u)$ and signature key pair $(\mathbf{k}_u^{\mathbf{ver}}, \mathbf{k}_u^{\mathbf{sig}}) \leftarrow \mathbf{Gen}^{\mathbf{Sig}}(u)$.
- Add $(u, \mathbf{k}_u^{\mathbf{enc}}, \mathbf{k}_u^{\mathbf{ver}})$ to USERS

$delU(u)$
- Delete $\mathbf{k}_u^{\mathbf{enc}}$ and $\mathbf{k}_u^{\mathbf{ver}}$
- For every role r that u is a member of:
  - $revokeU(u, r)$

$addP_u(fn, f)$
- Generate symmetric key $k \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$
- Send $\langle \mathsf{F}, fn, 1, \mathbf{Enc}_k^{\mathbf{Sym}}(f), \mathbf{Sign}_{\mathbf{k}_u^{\mathbf{sig}}}^{\mathbf{Sig}} \rangle$ and $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, \mathbf{Enc}_{SU}^{\mathbf{Pub}}(k), u, \mathbf{Sign}_{\mathbf{k}_u^{\mathbf{sig}}}^{\mathbf{Sig}} \rangle$ to R.M.
- The R.M. receives $\langle \mathsf{F}, fn, 1, c, sig \rangle$ and $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, c', u, sig' \rangle$ and verifies that the tuples are well-formed and the signatures are valid, i.e., $\mathbf{Ver}_u^{\mathbf{Sig}}(\langle \mathsf{F}, fn, 1, c \rangle, sig) = 1$ and $\mathbf{Ver}_u^{\mathbf{Sig}}(\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, c', u \rangle, sig') = 1$.
- If verification is successful, the R.M. adds $(fn, 1)$ to FILES and stores $\langle \mathsf{F}, fn, 1, c \rangle$ and $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, 1, c', u, sig' \rangle$

$delP(fn)$
- Remove (fn, v_{fn}) from FILES
- Delete $\langle \mathsf{F}, fn, -, - \rangle$ and all $\langle \mathsf{FK}, -, \langle fn, - \rangle, -, -, -, - \rangle$

$addR(r)$
- Generate encryption key pair $(\mathbf{k}_{(r,1)}^{\mathbf{enc}}, \mathbf{k}_{(r,1)}^{\mathbf{dec}}) \leftarrow \mathbf{Gen}^{\mathbf{Pub}}((r, 1))$ and signature key pair $(\mathbf{k}_{(r,1)}^{\mathbf{ver}}, \mathbf{k}_{(r,1)}^{\mathbf{sig}}) \leftarrow \mathbf{Gen}^{\mathbf{Sig}}((r, 1))$
- Add $(r, 1, \mathbf{k}_{(r,1)}^{\mathbf{enc}}, \mathbf{k}_{(r,1)}^{\mathbf{ver}})$ to ROLES
- Send $\langle \mathsf{RK}, SU, (r, 1), \mathbf{Enc}_{SU}^{\mathbf{Pub}}\left(\mathbf{k}_{(r,1)}^{\mathbf{dec}}, \mathbf{k}_{(r,1)}^{\mathbf{sig}}\right), \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.

$delR(r)$
- Remove $(r, v_r, -, -)$ from ROLES
- Delete all $\langle \mathsf{RK}, -, (r, v_r), -, - \rangle$
- For all permissions $p = \langle fn, op \rangle$ that r has access to:
  - $revokeP(r, \langle fn, \mathsf{RW} \rangle)$

$assignU(u, r)$
- Find $\langle \mathsf{RK}, SU, (r, v_r), c, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{RK}, SU, (r, v_r), c \rangle, sig) = 1$
- Decrypt keys $(\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}, \mathbf{k}_{(r,v_r)}^{\mathbf{sig}}) = \mathbf{Dec}_{SU}^{\mathbf{Pub}}(c)$
- Send $\langle \mathsf{RK}, u, (r, v_r), \mathbf{Enc}_{\mathbf{k}_u^{\mathbf{enc}}}^{\mathbf{Pub}}\left(\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}, \mathbf{k}_{(r,v_r)}^{\mathbf{sig}}\right), \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.

$revokeU(u, r)$
- Generate new role keys $(\mathbf{k}_{(r,v_r+1)}^{\mathbf{enc}}, \mathbf{k}_{(r,v_r+1)}^{\mathbf{dec}}) \leftarrow \mathbf{Gen}^{\mathbf{Pub}}((r, v_r+1))$, $(\mathbf{k}_{(r,v_r+1)}^{\mathbf{ver}}, \mathbf{k}_{(r,v_r+1)}^{\mathbf{sig}}) \leftarrow \mathbf{Gen}^{\mathbf{Sig}}((r, v_r+1))$
- For all $\langle \mathsf{RK}, u', (r, v_r), c, sig \rangle$ with $u' \neq u$ and $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{RK}, u', (r, v_r), c \rangle, sig) = 1$:
  - Send $\langle \mathsf{RK}, u', (r, v_r+1), \mathbf{Enc}_{\mathbf{k}_{u'}^{\mathbf{enc}}}^{\mathbf{Pub}}\left(\mathbf{k}_{(r,v_r+1)}^{\mathbf{dec}}, \mathbf{k}_{(r,v_r+1)}^{\mathbf{sig}}\right), \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.
- For every fn such that there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v_{fn}, c, SU, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), p, v_{fn}, c, SU \rangle, sig) = 1$:
  - For every $\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c', SU, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c', SU \rangle, sig) = 1$:
    - Decrypt key $k = \mathbf{Dec}_{\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}}^{\mathbf{Pub}}(c')$
    - Send $\langle \mathsf{FK}, (r, v_r + 1), \langle fn, op' \rangle, v, \mathbf{Enc}_{\mathbf{k}_{(r,v_r+1)}^{\mathbf{enc}}}^{\mathbf{Pub}}(k), SU, \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.
  - Generate new symmetric key $k' \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$ for p
  - For all $\langle \mathsf{FK}, id, \langle fn, op' \rangle, v_{fn}, c'', SU, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, id, \langle fn, op' \rangle, v_{fn}, c'', SU \rangle, sig) = 1$:
    - Send $\langle \mathsf{FK}, id, \langle fn, op' \rangle, v_{fn} + 1, \mathbf{Enc}_{id}^{\mathbf{Pub}}(k'), SU, \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.
  - Increment v_{fn} in FILES, i.e., set $v_{fn} := v_{fn} + 1$
- Update r in ROLES, i.e., replace $(r, v_r, \mathbf{k}_{(r,v_r)}^{\mathbf{enc}}, \mathbf{k}_{(r,v_r)}^{\mathbf{ver}})$ with $(r, v_r + 1, \mathbf{k}_{(r,v_r+1)}^{\mathbf{enc}}, \mathbf{k}_{(r,v_r+1)}^{\mathbf{ver}})$
- Delete all $\langle \mathsf{RK}, -, (r, v_r), -, - \rangle$
- Delete all $\langle \mathsf{FK}, (r, v_r), -, -, -, -, - \rangle$

$assignP(r, \langle fn, op \rangle)$
- For all $\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, v, c, id, sig \rangle$ with $\mathbf{Ver}_{id}^{\mathbf{Sig}}(\langle \mathsf{FK}, SU, \langle fn, \mathsf{RW} \rangle, v, c, id), sig) = 1$:
  - If this adds Write permission to existing Read permission, i.e., $op = \mathsf{RW}$ and there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{Read} \rangle, v, c', SU, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c', SU \rangle, sig) = 1$:
    - Send $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{RW} \rangle, v, c', SU, \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.
    - Delete $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{Read} \rangle, v, c', SU, sig \rangle$
  - If the role has no existing permission for the file, i.e., there does not exist $\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c', SU, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op' \rangle, v, c, SU \rangle, sig) = 1$:
    - Decrypt key $k = \mathbf{Dec}_{\mathbf{k}_{SU}^{\mathbf{dec}}}^{\mathbf{Pub}}(c)$
    - Send $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v, \mathbf{Enc}_{\mathbf{k}_{(r,v_r)}^{\mathbf{enc}}}^{\mathbf{Pub}}(k), SU, \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.

$revokeP(r, \langle fn, op \rangle)$
- If $op = \mathsf{Write}$:
  - For all $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{RW} \rangle, v, c, SU, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{RW} \rangle, v, c, SU \rangle, sig) = 1$:
    - Send $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{Read} \rangle, v, c, SU, \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.
    - Delete $\langle \mathsf{FK}, (r, v_r), \langle fn, \mathsf{RW} \rangle, v, c, SU, sig \rangle$
- If $op = \mathsf{RW}$:
  - Delete all $\langle \mathsf{FK}, (r, v_r), \langle fn, - \rangle, -, -, - \rangle$
  - Generate new symmetric key $k' \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$
  - For all $\langle \mathsf{FK}, id, \langle fn, op' \rangle, v_{fn}, c, SU, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, id, \langle fn, op' \rangle, v, c, SU \rangle, sig) = 1$:
    - Send $\langle \mathsf{FK}, id, \langle fn, op' \rangle, v_{fn} + 1, \mathbf{Enc}_{id}^{\mathbf{Pub}}(k'), SU, \mathbf{Sign}_{SU}^{\mathbf{Sig}} \rangle$ to R.M.
  - Increment v_{fn} in FILES, i.e., set $v_{fn} := v_{fn} + 1$

$read_u(fn)$
- Find $\langle \mathsf{F}, fn, v, c \rangle$ with valid ciphertext c
- Find a role r such that the following hold:
  - u is in role r, i.e., there exists $\langle \mathsf{RK}, u, (r, v_r), c', sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{RK}, u, (r, v_r), c' \rangle, sig) = 1$
  - r has read access to version v of fn, i.e., there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v, c'', SU, sig' \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v, c'', SU \rangle, sig') = 1$
- Decrypt role key $\mathbf{k}_{(r,v_r)}^{\mathbf{dec}} = \mathbf{Dec}_{\mathbf{k}_u^{\mathbf{dec}}}^{\mathbf{Pub}}(c')$
- Decrypt file key $k = \mathbf{Dec}_{\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}}^{\mathbf{Pub}}(c'')$
- Decrypt file $f = \mathbf{Dec}_k^{\mathbf{Sym}}(c)$

$write_u(fn, f)$
- Find a role r such that the following hold:
  - u is in role r, i.e., there exists $\langle \mathsf{RK}, u, (r, v_r), c, sig \rangle$ with $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{RK}, u, (r, v_r), c \rangle, sig) = 1$
  - r has write access to the newest version of fn, i.e., there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v_{fn}, c', SU, sig' \rangle$ and $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v, c', SU \rangle, sig') = 1$
- Decrypt role key $\mathbf{k}_{(r,v_r)}^{\mathbf{dec}} = \mathbf{Dec}_{\mathbf{k}_u^{\mathbf{dec}}}^{\mathbf{Pub}}(c)$
- Decrypt file key $k = \mathbf{Dec}_{\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}}^{\mathbf{Pub}}(c')$
- Send $\langle \mathsf{F}, fn, v_{fn}, \mathbf{Enc}_k^{\mathbf{Sym}}(f), \mathbf{Sign}_{(r,v_r)}^{\mathbf{Sig}} \rangle$ to R.M.
- The R.M. receives $\langle \mathsf{F}, fn, v, c'', sig'' \rangle$ and verifies the following:
  - The tuple is well-formed with $v = v_{fn}$
  - The signature is valid, i.e., $\mathbf{Ver}_{\mathbf{k}_{(r,v_r)}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{F}, fn, v, c'' \rangle, sig'') = 1$
  - r has write access to the newest version of fn, i.e., there exists $\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v_{fn}, c', SU, sig' \rangle$ and $\mathbf{Ver}_{\mathbf{k}_{SU}^{\mathbf{ver}}}^{\mathbf{Sig}}(\langle \mathsf{FK}, (r, v_r), \langle fn, op \rangle, v_{fn}, c', SU \rangle, sig') = 1$
- If verification is successful, the R.M. replaces $\langle \mathsf{F}, fn, -, - \rangle$ with $\langle \mathsf{F}, fn, v_{fn}, c'' \rangle$

Figure A.1: Implementation of RBAC_0 using PKI

Appendix B CryptoAC Screenshots

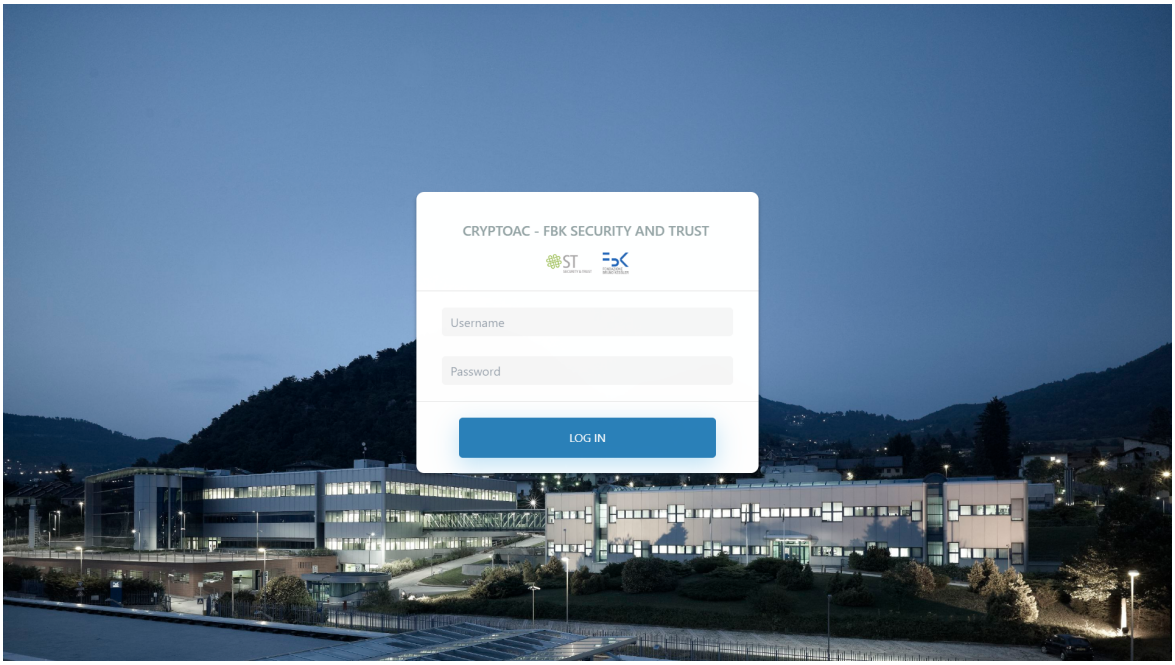


Figure B.1: Login Screen

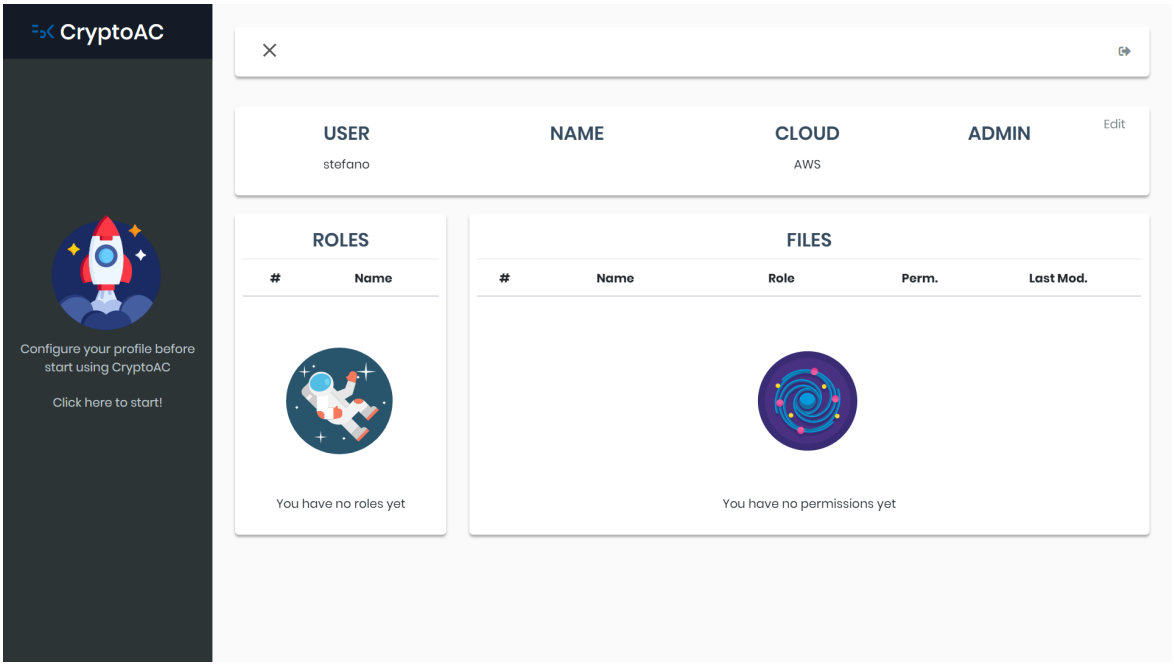


Figure B.2: Home Page

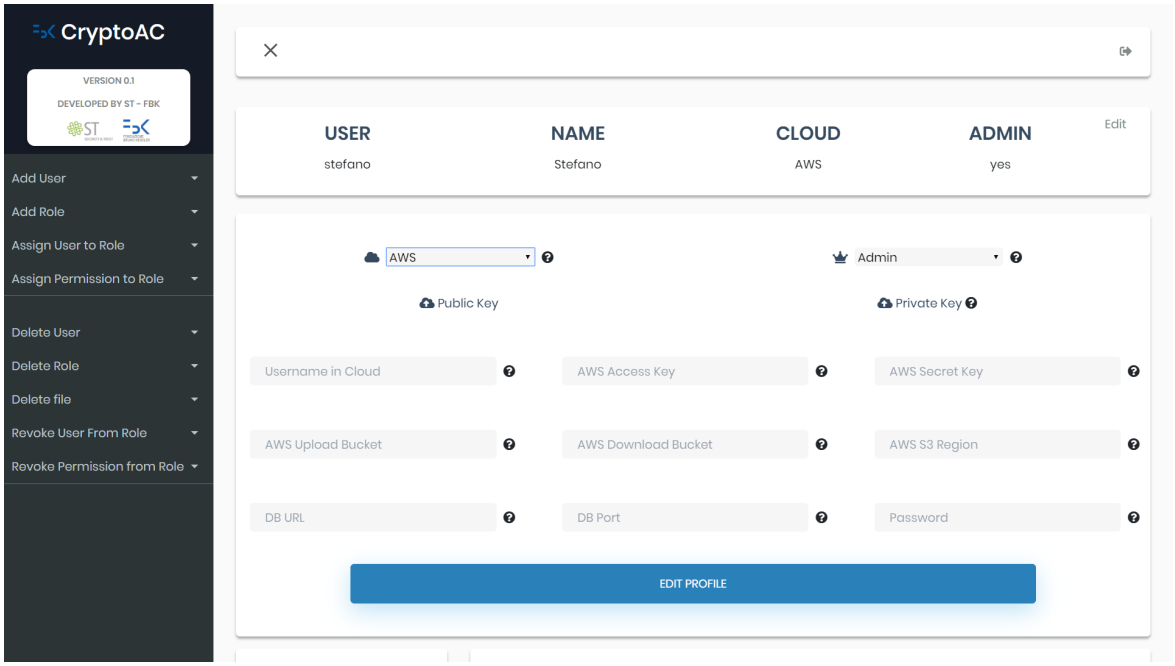


Figure B.3: Profile Settings

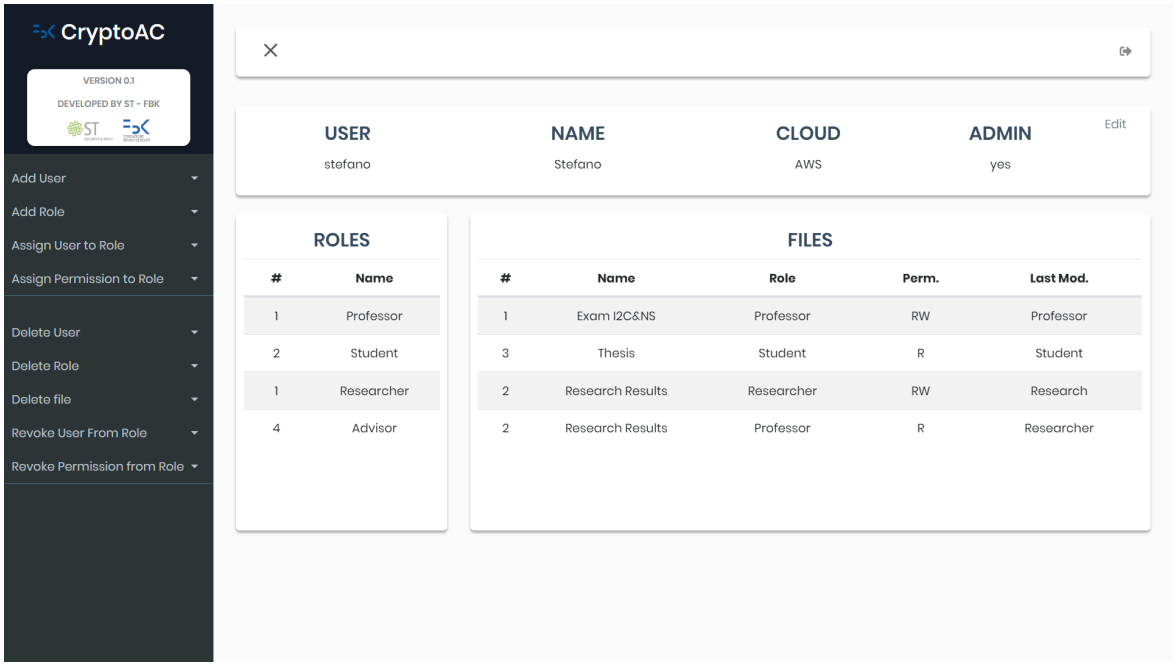


Figure B.4: Administrator's Dashboard