Exploring Architectures for Cryptographic Access Control Enforcement in the Cloud for Fun and Optimization

Stefano Berlato
Security and Trust Research Unit
Fondazione Bruno Kessler
Trento, Italy
sberlato@fbk.eu

Adam J. Lee
Computer Science Department
University of Pittsburgh
Pittsburgh, United States
adamlee@cs.pitt.edu

Roberto Carbone
Security and Trust Research Unit
Fondazione Bruno Kessler
Trento, Italy
carbone@fbk.eu

Silvio Ranise
Security and Trust Research Unit
Fondazione Bruno Kessler
Trento, Italy
ranise@fbk.eu

ABSTRACT

To facilitate the adoption of cloud by organizations, Cryptographic Access Control (CAC) is the obvious solution to control data sharing among users while preventing partially trusted Cloud Service Providers (CSP) from accessing sensitive data. Indeed, several CAC schemes have been proposed in the literature. Despite their differences, available solutions are based on a common set of entities—e.g., a data storage service or a proxy mediating the access of users to encrypted data—that operate in different (security) domains—e.g., on-premise or the CSP. However, the majority of the CAC schemes assume a fixed assignment of entities to domains; this has security and usability implications that are not made explicit and can make inappropriate the use of a CAC scheme in certain scenarios with specific requirements. For instance, assuming that the proxy runs at the premises of the organization avoids the vendor lock-in effect but may substantially undermine scalability.

To the best of our knowledge, no previous work considers how to select the best possible architecture (i.e., the assignment of entities to domains) to deploy a CAC scheme for the requirements of a given scenario. In this paper, we propose a methodology to assist administrators in exploring different architectures of CAC schemes for a given scenario. We do this by identifying the possible architectures underlying the CAC schemes available in the literature and formalizing them in simple set theory. This allows us to reduce the problem of selecting the most suitable architecture satisfying a heterogeneous set of requirements arising from the considered scenario to a Multi-Objective Optimization Problem (MOOP) for which state-of-the-art solvers can be invoked. Finally, we show how the capability of solving the MOOP can be used to build a prototype tool assisting administrators to preliminary perform a "What-if"

analysis to explore the trade-offs among the various architectures and then use available standards and tools (such as TOSCA and Cloudify) for automated deployment in multiple CSPs.

CCS CONCEPTS

• **Security and privacy** → **Access control**; *Cryptography*.

KEYWORDS

Cryptographic Access Control; Architecture; Optimization

1 INTRODUCTION

Cryptographic Access Control (CAC) allows organizations and users to enforce Access Control (AC) on cloud-hosted sensitive data while preserving data confidentiality with respect to both external attackers and the cloud itself. Several CAC schemes have been proposed in the literature, each embodying particular features through different cryptographic primitives. Among others, Attribute-Based Encryption (ABE) is employed by some CAC schemes [13, 18, 39] due to its ability to enforce rich Attribute-Based AC (ABAC) policies. Other schemes combine asymmetric and symmetric cryptography in hybrid cryptosystems [10], employ lazy revocation [40] or express other AC models like Role-Based AC [41] (RBAC). Others adopt proxy re-encryption [32] or onion encryption [29] to offload the burden of cryptographic operations to the cloud.

Problem Statement. While these CAC schemes offer advanced and remarkable features, they are often not suitable for a concrete use [10]. For instance, ABE applied to AC in the cloud "only exists in an academic world and it is often difficult to find a practical use of ABE for a real application" [18]. Since researchers often focus on high-level features only, little space is left for aspects related to the deployment of their scheme in a given scenario. An important aspect for the deployment of CAC schemes is the definition of the software entities that compose the CAC scheme along with

the entities' logical or physical locations (i.e., the definition of the "architecture" of the CAC scheme). However, CAC schemes are seldom provided with an architecture or this is usually fixed and it cannot adapt to the requirements (e.g., enhance architecture scalability or reduce architecture-related monetary costs) of different scenarios. Indeed, while CAC has been studied in several scenarios like eGovernment [15, 21] and eHealth [1, 7, 15, 25, 28, 32], we note that different scenarios have different requirements on the architectures of CAC schemes (i.e., architectural requirements). For instance, the eGovernment scenario may favour scalability and reliability of the architecture, while the eHealth scenario could require more control over the data. Unfortunately, the lack of study on the relationship between the architectures and the requirements of scenarios hampers the adoption of CAC schemes, since these are incapable of adapting to the mutable requirements of different scenarios. In other words, there is little or no research on how to fill the gap between CAC schemes in the abstract and an architecture for a deployment in a given scenario.

Solution. In this paper, we propose a methodology for finding the optimal architecture meeting the specific requirements of a given scenario for the enforcement of CAC schemes in the cloud. In particular, our contributions are as follows:

- we provide an architectural model to capture elements— namely resources, entities and domains— commonly involved in the architectures of CAC schemes. The architectural model formally expresses the set of the possible architectures of CAC schemes which preserve the expected confidentiality, integrity and availability properties of the involved resources. Then, to assess the model generality, we illustrate how the architectures of some state-of-the-art CAC schemes can be specified in our architectural model;
- we define how to evaluate different architectures based on security and usability goals that may be desirable in different scenarios. Then, we formalize a MOOP (which can be solved leveraging well-known techniques for Pareto optimality) to select the most suitable architecture that satisfies the goals of a specific scenario;
- we give a proof-of-concept application of how the architectural model and the MOOP can be used to assist administrators in the deployment of CAC schemes architectures. We develop a web dashboard[1] to solve a specific formalization of the optimization problem and perform a "What-if" analysis to further tune the requirements of the scenario and check the resulting architecture in real-time. To ensure cloud portability, interoperability and automatic deployment of the resulting architecture, we rely on the TOSCA (Topology and Orchestration Specification for Cloud Applications) OASIS standard to automatically generate a deployable specification of the resulting architecture. Finally, we implement a CAC scheme supporting such an architecture (i.e., the scheme proposed in [10]) and provide a fully working prototype with Amazon Web Services (AWS).

The paper is structured as follows. In Section 2 we introduce the background. In Section 3, we illustrate two important scenarios

often considered in the cloud-relevant literature, namely eGovernment and eHealth, while in Section 4 we introduce our architectural model. We present the optimization problem in Section 5 and our validation deployment in Section 6. In Section 7 we discuss related work and we conclude the paper with final remarks and future work in Section 8.

2 BACKGROUND

In this section, we introduce AC, RBAC and present the high-level functioning of a cryptographic RBAC scheme.

2.1 Access Control

Samarati and De Capitani di Vimercati [34] defined AC as "the process of mediating every request to resources maintained by a system and determining whether the request should be granted or denied". Resources usually consist of data such as files and documents. AC is traditionally divided into three levels:

- *Policy*: this abstract level consists of the rules stating which users can perform which operations on which resources. The policy is usually defined by the owner of the resources or of the system (e.g., the organization);
- *Model*: this intermediate level is a formal representation of the policy (e.g., RBAC [35] and ABAC [16] are two models) giving the semantics to granting or denying users' requests;
- *Enforcement*: this concrete level comprehends the hardware and software entities that enforce the policy based on the chosen model. The physical or logical location of these entities along with their interactions (i.e., the architecture) is part of the enforcement level.

These three levels are independent of each other. This allows evaluating different enforcement mechanisms for the same policy and model.

RBAC is one of the most widely adopted AC models in which *Users* are assigned to one or more *roles*. In the context of an organization, a *role* reflects an internal qualification (e.g., employee). *Permissions* are assigned to one or more *roles* by administrators of the policy. *Users* activate some *roles* to access the *permissions* needed to finalize their operations (e.g., read a file). Formally, the state of an RBAC policy can be described by the set of users U, roles R, permissions P and the assignments *users-roles* $UR \subseteq U \times R$ and *roles-permissions* $PA \subseteq R \times P$. A *user u* can use *permission p* if $\exists r : ((u, r) \in UR) \land ((r, p) \in PA)$. We note that *role* hierarchies can always be compiled away by adding suitable pairs to *UA*.

There are two main classes of enforcements for AC. In the first class, a trusted central entity decides whether to grant a specific action on a resource to a given user. All resources are stored in one or more trusted logical or physical locations (i.e., domains) to which the trusted entity has full access. Unfortunately, this trusted entity may not always be present in every scenario. Therefore, the second class studies the enforcement of AC policies in partially trusted domains [4, 9]. A partially trusted domain is a domain controlled by a third-party (e.g., an external organization or a cloud service provider) which faithfully performs the assigned instructions (e.g., store the data) but, at the same time, it tries to extract information from the stored data. If data are sensitive, this behaviour may be undesirable. A CSP is an example of a partially trusted domain,

---

[1]see https://stfbk.github.io/complementary/ASIACCS2020

as traditionally assumed in the literature of cloud computing [5]. Indeed, a report by the U.S. Federal Trade Commission [31] states that CSPs regularly collect companies' data without the latter's knowledge. When trust on the participant entities is limited, resources are often encrypted to ensure confidentiality (e.g., through encryption) and integrity (e.g., through signatures).

2.2 Cryptographic Access Control Schemes

In partially trusted environments, CAC is often used to enforce AC while ensuring the confidentiality of sensitive data. Data are encrypted and the *permission* to read the encrypted data is embodied by the secret decrypting keys. While implying a further computational burden (i.e., the cryptographic operations), CAC allows encrypted data to be stored in partially trusted domains.

For concreteness, we present the CAC scheme proposed in [10] for enforcing cryptographic RBAC policies, although our findings can be generalized for other CAC schemes (e.g., [28, 32, 39–41]). To abstract from low level details, we assume that all communications occur through pairwise-authenticated and private channels (e.g., TLS). In the proposed scheme, each *user u* and each *role r* is provided with a pair of secret and public keys $(\mathbf{k}_u^s, \mathbf{k}_u^p)$ and $(\mathbf{k}_r^s, \mathbf{k}_r^p)$, respectively. Each file is encrypted with a different symmetric key $\mathbf{k}^{sym}$. To assign a *user* to a *role*, the *role*'s secret key $\mathbf{k}_r^s$ is encrypted with $\mathbf{k}_u^p$, resulting in $\{\mathbf{k}_r^s\}_{\mathbf{k}_u^p}$. To give read *permission* to a *role*, the symmetric key $\mathbf{k}^{sym}$ is encrypted with $\mathbf{k}_r^p$, resulting in $\{\mathbf{k}^{sym}\}_{\mathbf{k}_r^p}$. The use of both secret-public and symmetric cryptographic schemes is usually called "Hybrid Encryption" [10]. The policy is enforced through the encrypted cryptographic keys and further auxiliary data (e.g., files version numbers and digital signatures), together referred to as metadata. Both encrypted data and metadata are stored in the cloud. To read a file, a *user* performs the following actions through a software entity usually called proxy:

(1) The *user* decrypts the *role*'s encrypted secret key $\{\mathbf{k}_r^s\}_{\mathbf{k}_u^p}$ with his secret key $\mathbf{k}_u^s$, obtaining $\mathbf{k}_r^s$;
(2) The *user* decrypts the encrypted symmetric key $\{\mathbf{k}^{sym}\}_{\mathbf{k}_r^p}$ with $\mathbf{k}_r^s$, obtaining $\mathbf{k}^{sym}$;
(3) The *user* decrypts the file with $\mathbf{k}^{sym}$.

To write on a file, a *user* performs the same operations to obtain the symmetric key $\mathbf{k}^{sym}$ and then sends the new (encrypted) file to the cloud. Finally, an entity in the cloud, usually called Reference Monitor (RM), checks whether the *user* has actually write *permission* before accepting the new file.

3 SCENARIOS AND PROBLEM STATEMENT

We study scenarios in which an organization outsources the storage of sensitive data to the cloud and wants to use a CAC scheme to preserve the data confidentiality in presence of partially trusted Cloud Service Provider (CSP). Besides the basic requirement of ensuring data confidentiality (i.e., besides enforcing the AC policy), each scenario has different architectural requirements (e.g., simplify maintenance or enhance reliability). We note that different CAC schemes architectures have a different impact on these requirements. For instance, the architecture of the CAC scheme presented in Section 2.2 assumes the data and the RM to stay in the cloud domain, while the proxy is installed in the computer of

each user. By using the cloud, this architecture gains scalability and reliability, but it may suffer from high cloud-related monetary costs and the negative "Vendor Lock-in" effect, i.e., the more cloud services are used, the more difficult is to switch to another cloud. Hosting the RM at the premises of the organization can partially relieve these issues but may give rise to other concerns (e.g., software maintenance effort and weakness to Denial-of-Service attacks). In this paper, we develop a tool-supported methodology that assists administrators in evaluating these kinds of trade-offs.

Preliminary, we present two scenarios often studied in the literature of CAC schemes, namely eGovernment and eHealth. We discuss their requirements and highlight the importance of carefully analyzing architectural trade-offs when deploying a CAC scheme.

3.1 eGovernment Scenario

The eGovernment scenario is getting more attention [15, 21] as the Public Administrations (PAs) in different countries (e.g., Italy[2], Spain[3]) start a digitalization process to simplify the maintenance of their infrastructure by using the cloud.[4]

Based on technologies that include mobile and web applications together with electronic identity services (besides cloud computing), PAs can develop and provide a portfolio of public services. Suppose a PA wants to allow citizens to access government-issued personal documents (e.g., tax certificates) from anywhere and anytime. A European citizen may use eIDAS[5] to authenticate in an online service of a foreign European country. Then, through a CAC scheme, the citizen may share his data (e.g., an electronic passport or the tax certificate) with a public authority of the foreign European country while still preserving end-to-end confidentiality [15]. We summarize some of the most important architectural requirements of the eGovernment scenario:

- eGR1 - enable citizens' access from anywhere and anytime;
- eGR2 - simplify the maintenance of the architecture;
- eGR3 - limit CSP-related costs for budget constraints;
- eGR4 - prioritize the scalability of the services.

It should be clear how difficult it is to select the most suitable CAC scheme architecture for the requirements of the eGovernment scenario, as this means finding the architecture that simultaneously maximizes the achievement of each requirement.

3.2 eHealth Scenario

The problem of storing medical data in the cloud has been widely studied in the literature [1] by many researchers [7, 15, 21, 25, 28, 32, 36, 40], along with the eHealth scenario requirements. For instance, Hörandner et al. [15] discussed the possible need for tracking patients' medical data from multiple devices (e.g., glucometers) continuously. These data are sent to the smartphone and finally encrypted and uploaded to the cloud. Domingo-Ferrer et al. [7] pointed out that, besides medical data (e.g., Blood sugar, LDL Cholesterol), also metadata should be hidden from the CSP, since they may leak sensitive information. Suppose a person with a mental disorder is

---

[2]https://www.agid.gov.it/it/infrastrutture/cloud-pa
[3]https://joinup.ec.europa.eu/collection/egovernment/news/spanish-government-approv
[4]https://joinup.ec.europa.eu/
[5]https://www.eid.as/home

hospitalized in a clinic specialized for treating mental disorders. The clinic is storing in the cloud the patients' medical data encrypted under a CAC scheme. However, suppose the CAC scheme expects the patient's name to be included in the metadata (e.g., in the AC policy or in the name of the file). Therefore, the CSP may infer that a specific person has a mental disorder. Consequently, the CSP may share this information for targeted advertisements or with a health insurance company that may then increase the insurance premium of the person. We summarize some of the most important architectural requirements of the eHealth scenario:

- eHR1 - hide metadata to avoid information leaking;
- eHR2 - prioritize redundancy to avoid medical data loss;
- eHR3 - limit the vendor lock-in effect.

As for the eGovernment scenario, it is not trivial how to find the architecture satisfying all or the highest number of these requirements at the same time.

3.3 Problem Characterization

Generalizing the scenarios presented in Section 3.1 and Section 3.2, we are interested in finding the CAC scheme architecture that strikes the best possible trade-off among the several requirements of a scenario. Depending on the scenario, there are many security and usability goals to consider and requirements to balance. Therefore, we argue that there is no single CAC scheme architecture that fits all scenarios. There is a need to carefully evaluate different architectures and find the one that maximizes the achievement of the requirements of each scenario. We do this by first formalizing the notion of CAC scheme architecture (Section 4) and then reducing the problem of finding the optimal architecture to a multi-objective optimization problem [22] (Section 5).

4 A MODEL FOR CAC ARCHITECTURES

While CAC schemes have different features, their architectures leverage several common elements (e.g., cryptographic keys, proxy, and reference monitor). We identify three sets to contain the basic building blocks of our model, namely (cryptographic) Resources, Domains and Entities. We also consider the set Properties to contain the three basic security properties, namely C(onfidentiality), I(ntegrity), and A(vailability). These sets are linked together by six relationships: domains can contain (*CanContain*) entities and preserve or not (*Preserves*) the security properties of resources, while entities use (*Uses*) and host (*Hosts*) resources and inherit (*Inherits*) security properties required (*Requires*) by resources. Figure 1a shows the situation as an Entity-Relation diagram where sets are depicted as rectangles with rounded corners and relations as diamonds.

Below, we define the three sets and six relations and explain how they are combined to specify an architecture for a CAC scheme. To show expressiveness and adequacy, we specify several architectures of CAC schemes proposed in the literature as instances of our architectural model. Formally, we work in basic set theory and use the standard notions of set membership ($\in$), containment ($\subseteq$), and set comprehension ($\{\cdot \mid \cdots\}$). Sometimes, we write $X(q)$ to denote $q \in X$ for q an element (a tuple) and X a set (relation, respectively). Figure 1 summarizes the concepts expressed in this section.
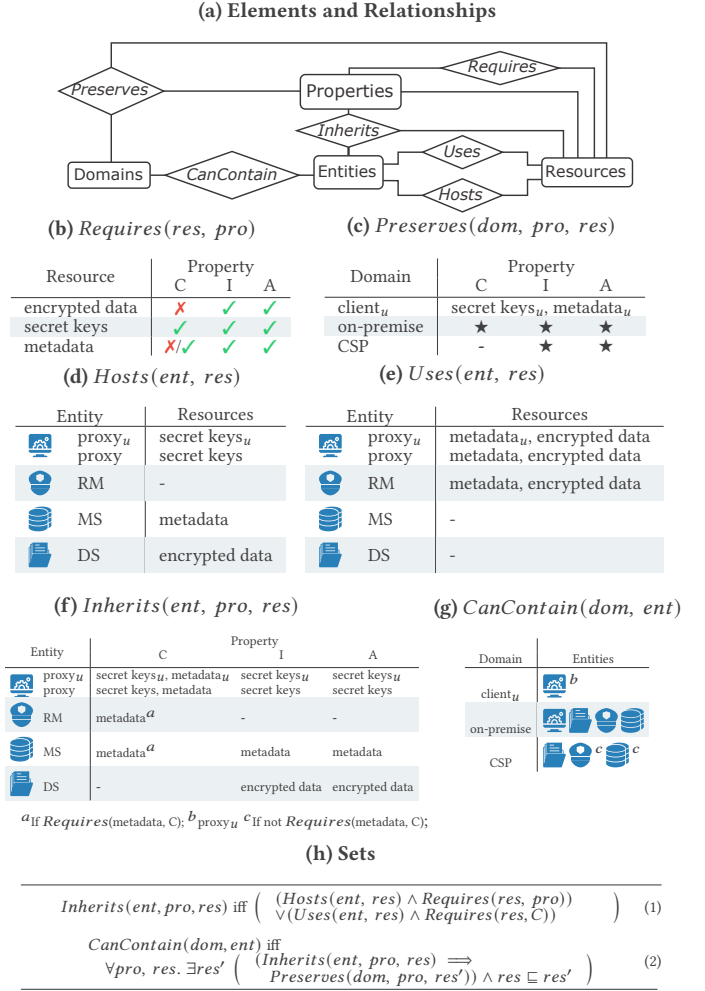

Figure 1: Summary of Basic Notions of CAC Schemes

4.1 Cryptographic Resources and Properties

The set Resources contains (cryptographic) resources of the following types (see the column Resource in Table 1b):

- *encrypted data*: by definition, the architecture of a CAC scheme involves data (e.g., files encrypted with $\mathbf{k^{sym}}$, as introduced in Section 2.2) encrypted under an AC policy;
- *secret keys*: a CAC scheme expects a set of secret cryptographic keys (e.g., the asymmetric keys of users and roles, $(\mathbf{k_u^s}, \mathbf{k_u^p})$ and $(\mathbf{k_r^s}, \mathbf{k_r^p})$, and the symmetric keys of files, $\mathbf{k^{sym}}$, as introduced in Section 2.2);
- *metadata*: intuitively, a CAC scheme needs also metadata (e.g., the AC policy, public cryptographic keys, files version numbers and digital signatures, as introduced in Section 2.2).

Since CAC schemes rely on these resources to properly function, we require to preserve their integrity (i.e., prevent unauthorized modifications) and availability (i.e., guarantee access when needed). However, a resource may be sensitive or not (e.g., public cryptographic keys are not sensitive). Therefore, we may require or not to preserve the Confidentiality of a (cryptographic) resource. Table 1b

defines the relation *Requires*, i.e., it identifies the CIA properties *pro* required by each resource *res*: $\langle res, pro \rangle \in$ *Requires* when the cell at the row *res* and column *pro* shows the symbol ✓ whereas $\langle \rangle res, pro \rangle \notin$ *Requires* when it contains the symbol ✗.

We assume perfect encryption over data (i.e., the confidentiality of the encrypted data cannot be compromised by attacking the available cryptographic primitives) so that confidentiality of encrypted data is implied. On the contrary, the confidentiality of the secret keys is crucial for the overall security of CAC schemes; we add this as a requirement. Therefore, we require to preserve the confidentiality of the keys. Finally, the sensitivity of the metadata depends on the organization and the scenario. For instance, the name of files can potentially disclose on what projects the organization is working, while the AC policy can reveal the organization's internal hierarchy [41]. Depending on the organization's judgment, metadata confidentiality can be either required or not (✓/✗). We note that sensitive metadata can be encrypted and turned into non-sensitive metadata at the cost of additional overhead on the CAC scheme. However, not all CAC schemes expect to encrypt metadata, and some entities may need to access plain-text metadata anyway. Therefore, we consider as optional the possibility to have sensitive metadata.

4.2 Domains and Trust

Following [10] and [40], we identify three domains defined from the organization's point of view. Domains are containers for other elements (e.g., a CSP hosting a database) and are grouped together in the set Domains (see the column Domain in Table 1c):

- $client_u$ is the domain in which the user u operates. We define the $client_u$ domain as the user's u personal devices (e.g., his laptop and smartphone). We assume that personal devices are not shared among users and that access is protected through passwords or similar authentication techniques. In this way, each user operates independently from the other users;
- *on-premise* is the domain in which the administrators operate. Usually, the on-premise domain lies within the organization as an area to which only authorized personnel can access (e.g., a data centre to which only administrators can access, either physically or virtually);
- *CSP* is the domain of a third-party offering cloud services, like computing and storage of files. It is a logical area and is geographically distributed [6].

The fact that a domain *dom* is assumed to preserve (or not) a CIA property *pro* of a resource *res* it contains is formalized as *Preserves*(*dom*, *pro*, *res*). We show in Table 1c the definition of the relation *Preserves*, where the symbol "★" is a wildcard for any resource and the symbol "-" stands for no resource. We consider administrators and thus the on-premise domain to be fully trusted. As a consequence, the on-premise domain preserves the CIA properties of all the resources it contains— formally, for *res* in Resources, *Preserves*(*on* − *premise*, *C*, *res*), *Preserves*(*on* − *premise*, *I*, *res*) as well as *Preserves*(*on* − *premise*, *A*, *res*). As discussed in Section 3, we assume the CSP to be partially trusted; this means that the CSP preserves the integrity and availability of the resources it contains but not the confidentiality—formally, *Preserves*(*CSP*, *I*, *res*),

Preserves(*CSP*, *A*, *res*), and (*CSP*, *C*, *res*) ∉ *Preserves* for *res* in Resources. Users are not trusted to operate on (i.e., they do not preserve the CIA properties of) resources the AC policy does not grant them access to. However, users are trusted to operate on resources the AC policy grants them access to (e.g., the user's own secret keys). To refer to the portion of a resource to which the user u has access to, we use the subscript u. For instance, secret keys$_u$ indicates the secret keys to which the user u has access to based on the AC policy (e.g., $(\mathbf{k}_u^s, \mathbf{k}_u^p)$) and not the whole set of secret keys (e.g., another user's u' keys, i.e., $(\mathbf{k}_{u'}^s, \mathbf{k}_{u'}^p)$). Similarly, metadata$_u$ refers to the portion of metadata the user u can access to based on the AC policy. Therefore, each client$_u$ domain preserves the CIA properties of the subset of resources the AC policy grants the user u access to (i.e., secret keys$_u$ and metadata$_u$). Formally, *Preserves*(*client$_u$*, *pro*, *secretkeys$_u$*) and *Preserves*(*client$_u$*, *pro*, *metadata$_u$*) for *pro* in Properties.

4.3 Entities and Relationships with Resources

The set Entities contains elements that actively perform tasks in CAC schemes (see the columns Entity in Table 1d and Table 1e):[6]

- *proxy*: Domingo-Ferrer et al. [7] argued that the architectures of CAC schemes usually involve a local proxy to interface users with encrypted data. The proxy takes care of encrypting the data before uploading them to the storage service and decrypting data before showing them to the user.
- *reference monitor* (RM): Garrison et al. [10] discussed the presence of a minimal reference monitor to check modifications to encrypted data. This entity checks the integrity and compliance with the AC policy of the users' actions (e.g., write on an encrypted file). Possibly, the RM also performs cryptographic operations (e.g., verifying digital signatures);
- *data storage* (DS): this entity is the storage (e.g., a database) containing the data;
- *metadata storage* (MS): this entity is the storage (e.g., a database) containing the metadata.

To accomplish its tasks, an entity must be located in at least one domain (e.g., a software needs to run on a machine); for this, an entity may host and use resources (e.g., a proxy using a secret cryptographic key to decrypt an encrypted file). Tables (d) and (e) define the relations *Hosts* and *Uses*, respectively, i.e., they identify the entity *ent* that hosts or uses a resource *res*. The proxy transforms high-level requests (e.g., read a file) into the sequence of low-level cryptographic operations necessary to accomplish them (e.g., obtain the decrypting key, download the encrypted file and decrypt the file, as presented in Section 2.2). Therefore, the proxy hosts the secret keys and uses metadata and encrypted data. We note that the proxy can be installed on each of the users' personal devices (i.e., multiple instances) or in a unique trusted location (i.e., single instance) like a server within the organization. In the former case, expressed as "proxy$_u$", each proxy hosts the secret keys of the user u and accesses metadata to which u has access to. In the latter case, expressed as "proxy", the proxy hosts the whole set of secret keys and accesses the whole set of metadata. Formally, we specify these as *Hosts*(*proxy*, *secret keys*), *Uses*(*proxy*, *metadata*), *Hosts*(*proxy$_u$*, *secret keys$_u$*), and *Uses*(*proxy$_u$*, *metadata$_u$*). Finally,

---

[6]Entities' icons made by Freepik from www.flaticon.com

the RM uses both metadata and encrypted data to verify the compliance with the AC policy of the users' actions; the DS and the MS store encrypted data and metadata, respectively. Formally, this is written as follows: $Hosts(MS, metadata)$, $Hosts(DS, encrypted\ data)$, $Uses(RM, metadata)$, and $Uses(RM, encrypted\ data)$.

4.4 Putting Things Together

We are now ready to define the notion of a CAC scheme architecture by identifying which CIA properties each entity inherits on which resources and then inferring in which domains an entity can stay by checking whether the domain preserves the properties inherited by the entity. To do this, we define two relations *Inherits* and *CanContain*, respectively. The intuition is that an architecture will be formed by those pairs $\langle ent, res \rangle$ that satisfies both relations for *ent* an element of Entities and *res* an element of Resources.

According to (1) in Table 1h, a tuple $\langle ent, pro, res \rangle \in Inherits$ if the entity *ent* hosts the resource *res* and *res* requires *pro*, or the entity *ent* uses the resource *res* and *res* requires *pro* having that *pro* is the confidentiality property. Table 1f is extensionally equivalent to (1) in Table 1h.

According to (2) in Table 1h, a tuple $\langle dom, ent \rangle \in CanContain$ if for all properties *pro* inherited by *ent* on *res*, *dom* preserves *pro* on res' with $res \sqsubseteq res'$ where $\sqsubseteq$ models a hierarchy on resources. The hierarchy refers to secret keys$_u$ $\sqsubseteq$ secret keys since secret keys$_u$ is a portion of secret keys and similarly metadata$_u$ $\sqsubseteq$ metadata as metadata$_u$ is a portion of metadata. Table 1g is extensionally equivalent to (2) in Table 1h and can be interpreted as follows. If we consider multiple instances of the proxy (a proxy$_u$ for each user u), the proxy would host a portion of the secret keys (i.e., secret keys$_u$) and access a portion of the metadata (i.e., metadata$_u$) only. In this case, the client$_u$ domain can contain the proxy$_u$. Then, being fully trusted by definition, the on-premise domain preserves the CIA properties of all resources and therefore can contain all entities. Finally, the CSP can contain the DS entity since the DS inherits the integrity and availability properties of the encrypted data only. Then, depending on the sensitivity of metadata, the RM and MS could inherit the confidentiality property. If the organization deems metadata not to be sensitive, the CSP can contain both the RM and MS. As a final note, since the RM provides security and compliance checks on users' actions, we assume that the RM cannot run in the users' computer. Therefore, we assume that $\langle client_u, RM \rangle \notin CanContain$.

As we can see from Table 1g, different domains can contain the same entity (e.g., both the on-premise and the CSP domain can contain the DS entity). It is important to notice that two or more domains can contain an entity at the same time. These hybrid architectures may be useful for entities hosting data (i.e., proxy, MS, DS). For instance, important encrypted files (e.g., with a sensitive name) can be hosted in a DS on-premise, while other files can stay in a DS in the cloud (e.g., [32]). The proxy can be installed in the computer of each user so to split the set of secret keys and also in an on-premise server to allow temporary users or light devices (e.g., smartphones) to access the architecture (e.g., [39]). The MS can be split so to host sensitive metadata (e.g., the list of users' names) on-premise and non-sensitive metadata (e.g., public cryptographic keys) in the CSP domain (e.g., [41]). In these hybrid architectures,

Table 2: Considered CAC Scheme Architectures

CAC Scheme	client$_u$	Architecture on-premise	CSP	*arc*
[10]				$\langle proxy_u, client_u \rangle, \langle MS, CSP \rangle, \langle RM, CSP \rangle, \langle DS, CSP \rangle$
[40]				$\langle proxy_u, client_u \rangle, \langle MS, CSP \rangle, \langle DS, CSP \rangle$
[41]				$\langle proxy_u, client_u \rangle, \langle MS, on\text{-}premise \rangle, \langle MS, CSP \rangle, \langle DS, CSP \rangle$
[39]				$\langle proxy_u, client_u \rangle, \langle proxy, on\text{-}premise \rangle,$ $\langle MS, on\text{-}premise \rangle, \langle RM, on\text{-}premise \rangle, \langle MS, CSP \rangle, \langle DS, CSP \rangle$
[28]				$\langle proxy, on\text{-}premise \rangle, \langle DS, on\text{-}premise \rangle, \langle DS, CSP \rangle$
[32]				$\langle proxy_u, client_u \rangle, \langle MS, on\text{-}premise \rangle,$ $\langle RM, on\text{-}premise \rangle, \langle DS, on\text{-}premise \rangle, \langle MS, CSP \rangle, \langle DS, CSP \rangle$

to avoid synchronization and update issues, we assume that each resource is hosted by one entity only. Of course, it is possible to have offline backups of the resources. Finally, we do not consider a hybrid architecture for the RM since it does not host any resource.

Architectural Model. By considering all possible entity-domain pairs that satisfies the constraints imposed by the *CanContain* relation (i.e., formally an architecture is a subset of the Cartesian product of the sets Entities and Domains), we identify 81 possible architectures for CAC schemes (see appendix A for the complete list). Each entity must be deployed in at least one of the domains that can contain it but the RM, that can be absent from the architecture as this happens in the architectures of some CAC schemes [3, 8, 12, 13, 24, 40]. In this case, after a `write` request, the old file is not replaced but a new version is added that is validated by the next user attempting to read the file. If the new version is not valid (i.e., the writer user did not have `write` permission), the reader fetches the old versions of the file until finding a valid version. Formally, we define the set $\mathcal{ARC}$ of all possible architectures *arc* as follows:

$$\mathcal{ARC} = \{arc \subseteq (\text{Entities} \times \text{Domains}) \mid (\forall \langle dom, ent \rangle \in arc : CanContain(dom, ent)) \land (\forall ent \in \text{Entities} \setminus \{RM\} \exists dom \in \text{Domains} : \langle dom, ent \rangle \in arc)\}; \quad (3)$$

4.5 Instances of our Architectural Model

Table 2 shows the architectures of some CAC schemes in the literature and how they are specified in our architectural model as elements of $\mathcal{ARC}$. We depict a hybrid architecture by duplicating the icon of the entity under multiple domains. We discuss how our model allows us to capture the most important aspects of the various CAC schemes in the following.

Garrison et al. [10] designed a CAC scheme for a dynamic RBAC policy with a focus on computational efficiency (e.g., hybrid encryption). The architecture comprehends the same three domains that we presented. A proxy for each user contains the user's secret keys. The (non-sensitive) metadata related to the AC policy are in the MS and the encrypted files in the DS entity. Both of these entities, together with the RM checking digital signatures on encrypted data, stay in the CSP domain.

In [40], the authors discussed the same three domains that we presented. The architecture of the CAC scheme expects a proxy for each user containing the user's secret keys (what the authors call "Key-store" module). Non-sensitive metadata (i.e., hierarchies

and cryptographic public parameters) are kept in the MS ("Metadata Directory" module) in the CSP domain. The DS ("Data Store" module) stores encrypted data in the CSP domain. As in [8], the authors proposed a CAC scheme without the RM, relying therefore on the users to validate `write` operations.

In [41], the authors employ Role-Based Encryption (RBE) to enforce RBAC policies in the CSP. In their architecture, the DS stores encrypted data in the CSP domain. Non-sensitive metadata (i.e., public parameters of RBE) are in the MS in the CSP domain, while sensitive metadata (i.e., role hierarchy and user memberships) stay in an MS within the organization. The architecture of the CAC scheme expects a proxy for each user. This CAC scheme does not support the `write` operation, thus the architecture does not expect the RM entity.

In [39], the authors proposed a CAC scheme along with a proof of concept prototype named "FADE". Users interact with a proxy ("FADE client") that can be deployed locally in each user's computer or as in a server within the organization. The architecture comprehends a quorum of key managers deployed as a centralized trusted entity within the organization. These key managers store sensitive metadata (e.g., cryptographic parameters) through threshold secret sharing [37]. The key managers perform blind decryption on cryptographic keys [27] and interact with the users to execute cryptographic operations during file uploads and downloads. Thus, the key managers act both as MS and RM. Encrypted files are stored by the DS in the CSP domain. Each file is associated with an AC policy (i.e., non-sensitive metadata) that is stored in the CSP.

Premarathne et al. [28] studied how to securely store medical big data in the cloud. They designed a role-based CAC scheme making use also of steganography. The architecture comprehends the "User" (i.e., client$_u$), "Health Authority" (i.e., on-premise), and "Cloud Storage" (i.e., CSP) domains. Users authenticate to a trusted health authority server. This server is responsible for extracting users' data (i.e., proxy) from files stored by the DS in the CSP domain. Metadata related to steganography (e.g., indexes and lengths) are stored in the health authority server (i.e., metadata storage). In this CAC scheme, the RM entity is missing. Indeed, since the proxy runs in the trusted on-premise domain, no one can tamper with it and proxy's actions are assumed to be legitimate.

In [32], the authors propose a CAC scheme based on a hybrid architecture. A private DS (i.e., on-premise domain) stores confidential patients' data (e.g., chronic diseases, mental health issues) and it can be accessed by authorized personnel only. A public DS (i.e., CSP domain) handles patient's data that are shared with other parties like medical researchers and government authorities. Access to the DSs is regulated by an RBAC policy. Therefore, each cloud has part of the metadata needed by the CAC scheme. Each user (e.g., doctors and nurses) is given secret keys.

5 TRADE-OFF ANALYSIS FOR ARCHITECTURAL DESIGNS

Once defined the set $\mathcal{ARC}$ of the possible architectures for CAC schemes, we formalize the problem (introduced in Section 3.3) of selecting the architecture that maximizes the achievement of multiple goals of a scenario as a MOOP [22]. Below, we first identify security and usability goals that may be desirable in different scenarios (as

described in Section 3.1 and Section 3.2). The set of goals is not meant to be exhaustive or representative, it is only given as an example to illustrate the optimization problem; other goals may easily be added. Then, we discuss the effect on the security and usability goals of different architectural choices. Finally, we show how to reduce the problem of selecting the architecture that maximizes the achievement of the desired goals into an optimization problem that considers the simultaneous maximization of a collection of objective functions that measure how much goals are achieved.

5.1 Identifying Goals

From cloud-relevant literature, we sample 8 security and usability goals that may be desirable in our scenarios:

- *Redundancy* [17, 20, 38]: the extent to which the architecture allows to effectively have duplicated resources;
- *Scalability* [19, 20, 28, 36, 39, 40]: the ability of the architecture to scale up and down to accommodate dynamic workloads (e.g., the variable number of users' requests or cryptographic operations);
- *Reliability* [17, 19, 20, 30, 36, 39]: the ability of the architecture to keep working after the failure of one or more entities. We measure reliability by considering Single-Point-of-Failures (SPOF);
- *Maintenance* [17, 20, 28, 36, 38]: the easiness in the deployment and maintenance (i.e., software updates) of the architecture;
- *Denial-of-Service Resilience* [10, 19, 20, 30]: the intrinsic resilience of the architecture to Denial-of-Service (DoS) attacks;
- *Minimization of CSP Vendor Lock-in* [19, 39]: the easiness in switching CSP in the architecture (e.g., from AWS to Azure);
- *On-premise Monetary Savings* [17, 19, 20, 28, 39]: the monetary savings due to *not* adding entities to the on-premise domain;
- *CSP Monetary Savings* [17, 19, 20, 28, 39]: the monetary savings due to *not* adding entities to the CSP domain. If the organization already has an internal infrastructure, it might be cheaper to run entities on-premise rather than in the CSP domain.

5.2 Effect on Goals

We adopt a modular approach to evaluate the effect that an architecture has on the various goals. We consider the effect of each entity—when contained in a certain domain—on each goal in isolation. We summarize our considerations in Table 3 by discussing the effects on the goals identified in Section 5.1. Each entity-domain pair may have either a positive (+), negligible (=) or negative (-) effect on a goal.

- *Effect on Redundancy*. The CSP is a geographically sparse domain with mechanisms for replicating data across large geographic distances [6]. On the contrary, the on-premise domain is, by definition, limited to one location. Therefore, redundancy is enhanced when entities are in the CSP domain;
- *Effect on Scalability*. As with the redundancy goal, scalability is a peculiarity of CSPs [6]. The more entities are in the

Table 3: Single Entity Effect on Goals

Goals	client$_u$	hybrid	on-premise	on-premise	CSP	on-premise	hybrid	CSP	on-premise	hybrid	CSP
Redundancy	=	=	=	-	+	-	=	+	-	=	+
Scalability	+	=	-	-	+	-	=	+	-	=	+
Reliability	+	=	-	-	+	-	=	+	-	=	+
Maintenance	+	=	-	-	+	-	=	+	-	=	+
DoS Resilience	+	=	-	-	+	-	=	+	-	=	+
Vendor Lock-in	=	=	=	+	-	+	=	-	+	=	-
On-premise Savings	+	=	-	-	+	-	=	+	-	=	+
CSP Savings	=	=	=	+	-	+	=	-	+	=	-

CSP domain, the more scalable is the architecture. Also, the architecture gets more scalable when the proxy is deployed in the client$_u$ domain, thus the burden of cryptographic operations is distributed among the users;

- *Effect on Reliability.* As for the redundancy goal, the CSP is generally more reliable than the on-premise domain [6]. Entities deployed in the on-premise domain create SPOFs and make the whole architecture less robust;

- *Effect on Maintenance.* The presence of entities in the on-premise domain leads to greater deployment (e.g., setup and configuration of the infrastructure) and maintenance (e.g., operative systems and runtime environments updates) effort. These issues are delegated to a third-party when entities are deployed in the CSP domain;

- *Effect on DoS Resilience.* We consider the CSP domain as intrinsically resistant to DoS attacks [6]. Therefore, the more entities are in the CSP (or client$_u$) domain, the more the architecture is DoS resistant. On the contrary, DoS attacks affect the availability of on-premise entities more easily;

- *Effect on Minimization of CSP Vendor Lock-in.* Intuitively, each entity in the CSP stresses the vendor lock-in effect. On the contrary, vendor lock-in is minimized when entities are in the on-premise and client$_u$ domains;

- *Effect on On-premise Monetary Savings.* The less the organization runs entities internally, the more the on-premise-related costs are reduced;

- *Effect on CSP Monetary Savings.* The less the organization deploys entities in the CSP, the more CSP-related costs are reduced.

From Table 3, we see that using the CSP yields advantages on several goals. This favours the use of the CSP in the architectures of CAC schemes. Indeed, CAC may be unnecessary in architectures not using the CSP, as resources would be stored in the trusted on-premise domain. In general, any architecture expecting the proxy in the on-premise domain may not use CAC, as a trusted proxy "would obviate the need for any cryptography beyond authenticated symmetric key encryption" [10].

In contrast, hybrid architectures tend to balance the pros and cons of the goals. For instance, assume that an architecture expects the MS to stay in the CSP domain and that the storage service is billed based on the amount of data stored (e.g., like AWS S3 pricing)[7]. In a hybrid architecture, metadata are split and stored in two MSs, one MS in the on-premise domain and one MS in the CSP

---
[7]https://aws.amazon.com/s3/pricing/

domain. Supposedly, the MS in the CSP domain would store only half of the metadata, resulting in half of the price (i.e., half of the savings). Therefore, we assume that hybrid architectures do not have an effect on the goals. This is just an example and the effect of hybrid architectures, as well as the others, can be tuned depending on the specific scenario and organization. In other words, it is up to the organization to tune the effects in Table 3 based on its specific needs.

5.3 Multi-Objective Optimization Problem

We consider a simple approach to combine the effect of each pair $\langle ent, dom \rangle$ on a goal, under the assumption that the effects are independent of each other. In practice, we "add" together the +, - and = symbols as adding the numbers +1, −1, and 0, respectively. Formally, this is equivalent to consider an objective function $g : \mathcal{ARC} \mapsto \mathbb{Z}$ associated with each goal. Having as input the set of $\langle ent, dom \rangle$ pairs of an architecture $arc \in \mathcal{ARC}$, the objective function returns the sum $g(arc)$ of the symbols (+, - and =) associated to each $\langle ent, dom \rangle$ pair, as defined in each row of Table 3. Note that hybrid architectures have two $\langle ent, dom \rangle$ pairs for the same ent.

Pre-filters. We note that not all architectures $arc \in \mathcal{ARC}$ may be of interest for a particular scenario. For instance, as said in Section 4.4, the architectures of some CAC schemes do not expect the RM. In this case, we should exclude from $\mathcal{ARC}$ every architecture expecting the presence of the RM. Furthermore, depending on the organization and scenario, there may or not be sensitive metadata. Architectures associating sensitive metadata with a domain not guaranteeing metadata confidentiality should also be excluded. For instance, an RM using or an MS hosting sensitive metadata cannot stay in the CSP domain. To formalize this, we define the set *Pre-Filters* containing the architectures that shall be excluded when finding the "optimal" architecture. Consequently, we define $\mathcal{ARC}_{\text{sub}} \subseteq \mathcal{ARC}$ as the set of architectures in $\mathcal{ARC}$ but not in *Pre-Filters*:

$$\mathcal{ARC}_{\text{sub}} = \{ arc \in \mathcal{ARC} | arc \notin \textit{Pre-Filters} \}; \qquad (4)$$

We are now ready to formalize as a MOOP [22] the problem of finding the optimal architecture $arc^* \in \mathcal{ARC}_{\text{sub}}$ such that the tuple $(g_{\text{Redundancy}}, \ldots, g_{\text{CSP Savings}})$ of objective goals measuring the degree of achievement of a goal is optimal:

$$\max_{arc \in \mathcal{ARC}_{\text{sub}}} (g_{\text{Redundancy}}(arc), \ldots, g_{\text{CSP Savings}}(arc)); \qquad (5)$$

Pareto Optimality. It is well-known that solving a MOOP is far from being trivial. The main source of difficulty is that it may be impossible to find a solution (an architecture, in our case) that simultaneously maximizes all objective functions. In fact, for any non-trivial MOOP, there is no single solution that is simultaneously optimal for every goal. Instead, there may exist many solutions that can be considered equally good, called Pareto optimal [22]. In our context, an architecture $arc^* \in \mathcal{ARC}_{\text{sub}}$ is *Pareto optimal* if and only if there is no architecture $arc \in \mathcal{ARC}_{\text{sub}}$ such that $g_i(arc) \geq g_i(arc^*)$ for each i in the set of goals (Redundancy, …, CSP Savings) and $g_j(arc) > g_j(arc^*)$ for at least one j in the set of goals. In other words, an architecture is Pareto optimal if there does not exist another architecture that improves one objective function without detriment to another. A crucial advantage of reducing
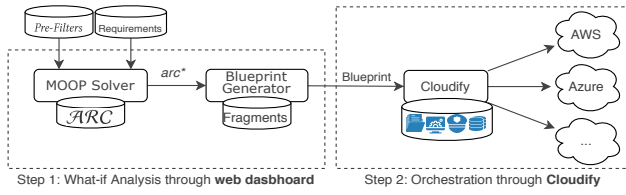
Figure 2: Two-step Deployment Process

the problem of selecting the most suitable architecture for a CAC scheme to a MOOP is the possibility to exploit out-of-the-shelf the several methods and tools that have been devised to help the process of choosing one or more solutions among those that are Pareto optimal. Many of these methods consist in transforming a MOOP into one or more (single goal) optimization problems whose solutions are Pareto optimal (under reasonable additional assumptions). We illustrate one such method in the following section.

6 ASSISTED DEPLOYMENT OF CAC SCHEMES

We now present how our architectural model and the MOOP in formula (5) can be used to assist administrators in the deployment of CAC schemes architectures. The workflow is summarized in Figure 2 as a two-step deployment process. The idea is to provide administrators with a web dashboard in which they can input the *Pre-Filters* set and the requirements of their scenario. In the first step, the web dashboard allows performing a thorough "What-if" analysis with the help of an automated MOOP solver to carefully assess the trade-offs of the security and usability goals among the Pareto optimal architectures in $\mathcal{ARC}_{\text{sub}}$. Through this analysis, the administrators can find the most suitable architecture $arc^* \in \mathcal{ARC}_{\text{sub}}$ for their scenario. Then, we automatically generate a deployable specification (called Blueprint) of arc^* based on the TOSCA[8] OASIS standard; a database containing blueprint fragments (Fragments) is used by the Blueprint Generator to build a TOSCA compliant representation of the arc^*. In the second step, we rely on the TOSCA-based Cloudify framework to automatically deploy the generated blueprint in the major CSPs, using the CAC entities we implemented (the blue icons). The ultimate purpose is to optimize and simplify the time consuming and error-prone activity of manually selecting and deploying CAC security schemes in the cloud.

Below, we describe a proof-of-concept application of the two-step deployment process for the eGovernment and eHealth scenarios. As a validation example, we formalize the MOOP into a single objective optimization problem (Section 6.1). Note that other formalizations are possible. Then, we input in the dashboard the requirements of the eGovernment (Section 6.1) and eHealth (Section 6.1) scenarios. We report the Blueprint fragments for the architecture arc^* of the eGovernment scenario in Appendix B. Finally, we describe the use of TOSCA and Cloudify to deploy the architecture arc^* for the eGovernment scenario (Section 6.2) and briefly discuss the implementation of a CAC scheme supporting this architecture (Section 6.3).

---

[8]https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

6.1 Constrained Weighted Sum Optimization

As a validation example, we choose to translate our MOOP into a simple objective optimization problem. We note that this is one possible formalization for our MOOP, where also others are possible (Minimum-Cost Flow [14] or Generalized Assignment Optimization Problem [26]) The objective function is the weighted sum of all the objective functions. In other words, we construct an objective function of the form $\sum_{i \in O} w_i \cdot g_i$, where O is the set of goals. The constants $w_i \in \mathbb{R}$'s are called *weights* and model the importance of achieving a certain goal. Technically, it is necessary to assume $w_i > 0$ for each i for guaranteeing that the solution of the transformed problem belongs to the set of Pareto optimal solutions of the original problem [22].

In some cases, a constrained variant of the weighted sum problem may be of interest. From the descriptions in Section 3.1 and Section 3.2, the eGovernment and eHealth scenarios may benefit from the enforcement of hard and soft constraints. These are respectively mandatory and optional thresholds values expressing conditions on the objective functions. In other words, if $g(arc)$ is less than the threshold value of a hard constraint, then the architecture arc is excluded from $\mathcal{ARC}_{\text{sub}}$, i.e., $arc \in$ *Pre-Filters*. Instead, if $g(arc)$ is less than the threshold value of a soft constraint, then $g(arc)$ is given a penalty, i.e., a penalty value $p \in \mathbb{Z}$ is subtracted from $g(arc)$. Below, we present two concrete applications of the weighted sum optimization problem to the eGovernment and eHealth scenarios.

eGovernment Scenario Application. As presented in Section 3.1, we consider a PA that wants to allow citizens to access government-issued personal documents (e.g., tax certificates) anywhere and anytime. We report in Figure 3 a screenshot of the web dashboard configured with the requirements of the eGovernment scenario.

The first black section allows to exclude specific $\langle ent, dom \rangle$ pairs from the architectures by toggling the visibility of entities' icons. This defines the starting subset of the architectures to consider in the optimization problem $\mathcal{ARC}_{\text{sub}} \subseteq \mathcal{ARC}$. For instance, the requirement eGR1 (enable citizens access from anywhere and anytime) may suggest storing data in the cloud to allow for ubiquitous access to encrypted data. This implies to remove the $\langle DS, on\text{-}premise \rangle$ pair by toggling the visibility of the DS under the on-premise domain. The second black section applies weights $w_i \in \mathbb{R}$ and constraints on the goals. For instance, the requirement eGR2 (simplify the maintenance of the architecture) can be translated as a hard constraint that excludes architectures with a negative value on the maintenance goal. Instead, the requirement eGR3 (limit CSP-related costs for budget constraints) can be translated as a soft constraint imposing a penalty (e.g., -5) for architectures with a negative value on the CSP monetary savings goal. The requirement eGR4 (prioritize the scalability of the services) can assign a weight to the scalability and reliability goals. For instance, the values can weight twice as much (i.e., $w_{\text{Scalability}} = 2$, $w_{\text{Reliability}} = 2$). The dashboard solves the constrained weighted sum optimization problem and shows in real-time, in the last two blue sections, the three most suitable architectures and the detail on the goals values, respectively. We see that the most suitable architecture uses CSP services as much as possible. This reflects the scalability and reliability priorities. At the same time, this architecture enhances the easiness in deployment and maintenance of the architecture.

Unacceptable <entity-domain> Pairs

Client	On-premise	CSP

Best Architectures

Client	On-premise	CSP

Constrained Weighted Sum Optimization Problem

	Redundancy	Scalability	Reliability	Maintenance	DoS Resilience	Vendor Lock-in	On-premise Savings	CSP Savings
Weight	1	2	2	1	1	1	1	1
Threshold	0	0	0	0	0	0	0	0
	○ Hard ○ Soft	○ Hard ○ Soft	○ Hard ○ Soft	⦿ Hard ○ Soft	○ Hard ○ Soft	○ Hard ○ Soft	○ Hard ○ Soft	○ Hard ⦿ Soft
Penalty	0	0	0	0	0	0	0	-5

Goals Analysis

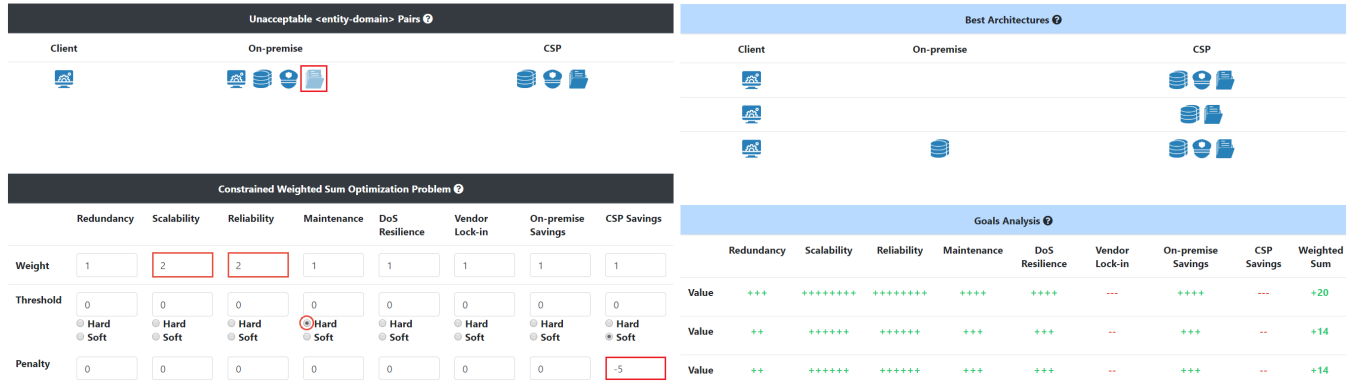|	Redundancy	Scalability	Reliability	Maintenance	DoS Resilience	Vendor Lock-in	On-premise Savings	CSP Savings	Weighted Sum
Value	+++	++++++++	++++++++	++++	++++	---	++++	---	+20
Value	++	++++++	++++++	+++	+++	--	+++	--	+14
Value	++	++++++	++++++	+++	+++	--	+++	--	+14

Figure 3: Web Dashboard Optimization Problem for eGovernment Scenario

By tuning (e.g., increasing) the penalty value on the CSP monetary savings goal, the weighted sum of these three architectures would decrease and other architectures may become optimal.

eHealth Scenario. We briefly discuss the scenario presented in Section 3. Suppose the presence of a clinic treating mental disorders. The requirement eHR1 (hide metadata to avoid information leaking) adds ⟨MS, CSP⟩ and ⟨RM, CSP⟩ to the *Pre-Filters* set. As said in Section 4.1, metadata could be encrypted and turned into not-sensitive metadata at the cost of additional overhead on the CAC scheme. This would allow the MS to stay in CSP domain. The RM, however, would still need to access plain-text metadata, so the RM cannot stay in the CSP domain anyway. The priority on the redundancy goal in eHR2 (prioritize redundancy to avoid medical data loss) translates in a weight applied to the value of that goal. For instance, the redundancy value can weight twice as much (i.e., $w_{Redundancy} = 2$). Finally, eHR3 (limit the vendor lock-in effect) can be seen as a hard constraint imposing a non-negative value on the vendor lock-in goal.

6.2 Architecture Modeling with Cloudify

Once identified the most suitable CAC scheme architecture for the eGovernment scenario, we need a CSP-independent modelling. We rely on the TOSCA OASIS standard for a flexible and portable representation of the architecture. TOSCA is a YAML-based modelling language addressing the lack of a standardized view on cloud services (e.g., storage, cloud functions). The goal of TOSCA is to allow one to migrate cloud applications across different CSP without investing high cost and time. For the actual modelling, we choose Cloudify,[9] a TOSCA-based open-source cloud orchestration framework supporting the major CSPs (e.g., Azure, AWS, Google, OpenStack). Cloudify allows graphical modeling of cloud applications by creating and configuring cloud services like servers and network appliances. The graphical model is called "blueprint" and it is composed of nodes representing cloud services (e.g., security groups, cloud functions) and relationships (e.g., a database hosted by a server). Given a blueprint, Cloudify automatically deploys and orchestrates the cloud application. We manually develop the blueprint templates of the most suitable CAC scheme architecture of

the eGovernment scenario and report it in Appendix B. In detail, we modelled an AWS relational database service (i.e., MS), a Lambda cloud function (i.e., RM) and the S3 storage service (i.e., DS). The proxy is not part of the blueprint since it is expected to be installed in users' computers.

6.3 CAC Scheme Implementation

As the last step, we provide a fully working implementation of the CAC scheme proposed in [10]. We choose this scheme because it supports the architecture of the eGovernment scenario. As explained in Section 2.2, the CAC scheme in [10] uses hybrid encryption. We choose RSA [33] for asymmetric encryption and AES for symmetric encryption. We implement the proxy as a Java program to be installed in the computer of each user. The users' secret key is generated and stored inside the user's proxy. We use AWS Java library to interface with AWS services and implement the Lambda cloud function code (i.e., RM). Finally, we test our implementation with several simulated sequences of operations. This means combining the creation of users and roles, assignment and revoking of permissions and the creation, update and management of files. We also implemented a graphical user interface based on web technologies. We made open-source the implementation of the scheme along with other resources.[10]

7 RELATED WORK

Cryptographic Access Control. CAC has been applied in several scenarios, like local filesystems [13] and the cloud [10]. Goyal et al. [13] developed a CAC scheme based on ABE. In their scheme, users can delegate their permissions but not revoke them. This makes the whole scheme not usable for a dynamic scenario. In [23], the authors proposed a similar scheme while also avoiding the disclosure of the AC policy itself, deemed to be sensitive metadata. Still, run-time modifications of the policy were not addressed. In [2], the authors considered the revocation of permissions, but they did not discuss the computational burden that a revocation implies. Garrison et al. [10] studied the computational usability of a simple dynamic Role-based CAC scheme. They concluded that, even when considering a minimally dynamic scenario, the CAC scheme is

---

[9]https://cloudify.co/

[10]see https://stfbk.github.io/complementary/ASIACCS2020

likely to produce significant computational overheads. Besides, many other security and usability goals are often overlooked when designing a CAC scheme, like scalability, reliability and monetary costs. Mainly, this is because a concrete deployment for a given scenario is seldom considered.

Cryptographic Access Control Architectures. There are few works [11, 18, 29, 36, 39–41] that presented an architecture for the CAC scheme they proposed.

In [36], the authors developed a scheme to allow multiple owners to give access to their data to multiple users, following a mixed Attribute-Role based CAC scheme. The architecture is composed of four modules responsible for users' authentication, AC policy management and data encryption. The authors discussed scalability and performance in `read` and `write` operations. However, the authors did not evaluate other security and usability goals of the architecture, nor they discussed the way the four modules should be deployed by the organization or how they should interact with the CSP. Also, they did not provide alternative designs for the architecture. The set up of many modules may carry considerable overhead, making the whole scheme unappealing for small companies.

In [41], the authors employ Role-Based Encryption to cryptographically enforce RBAC policies in the cloud. In their architecture, a CSP stores encrypted data while sensitive metadata are stored on-premise. Users communicate only with the CSP. In their architecture, the CSP is supposed to run cryptographic operations on behalf of the users and to communicate with the on-premise domain to retrieve the needed metadata. However, the authors did not discuss the feasibility of this communication, nor they analyzed other goals of the architecture. The authors developed a concrete implementation of their CAC scheme just for analyzing the performance of the `read` and `write` operations.

In [40], the authors proposed a CAC scheme emphasizing users' privacy by enabling anonymous access to resources stored in the CSP. The authors implemented a prototype interacting with AWS, providing an interface so that further CSPs can be supported. However, security and usability goals were not considered. The architecture is fixed and cannot be modified to accommodate different scenarios.

In [39], the authors proposed a CAC scheme and implemented it in a proof-of-concept prototype, named "FADE". The architecture comprehends a quorum of key managers deployed on-premise. Users interact with a FADE client that can be run in the users' devices or on-premise. 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 ABAC model, but no concrete design is given for such an extension.

Ghita et al. [11] implemented a cryptographic CAC scheme using ABE. Even though they developed a working prototype, many aspects were overlooked. For instance, in their CAC scheme it is not possible to add roles to the AC policy. This is a tight limitation on the usability of the scheme. The architecture is fixed and forces the proxy to run in the on-premise domain.

In [29], the authors propose a CAC scheme similar to the one presented by Garrison et al. [10]. However, revocation is handled through onion encryption. Each time a permission is revoked, the CSP adds an encryption layer with a new symmetric cryptographic key on each affected file. For reading a file, an authorized user has to decrypt each encryption layer. The administrator can set a threshold to the number of encryption layers on files, after which a de-onioning procedure occurs. The authors implemented their scheme and demonstrated how they obtain only slightly worse performances with respect to [10] (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 yields. Moreover, metadata are necessarily stored in the CSP. Unfortunately, the architecture was never taken into account explicitly. Again, the implementation had the only purpose of measuring the computational efficiency of the CAC scheme.

In [18], the authors designed and implemented a CAC scheme based on non-monotonic ciphertext-policy ABE. An administrator is responsible for creating cryptographic keys from users' attributes and each user has a proxy interacting with the CSP. Unfortunately, the authors used a programming library that not portable to other platforms. Most importantly, the CAC 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 CAC schemes with novel high-level features. Because of this, little space is left for additional analysis on security and usability goals or alternative architectures responsive to the requirements of a given scenario.

8 CONCLUSION AND FUTURE DIRECTIONS

In this paper, we proposed a methodology to find the most suitable architecture of CAC schemes for the requirements of different scenarios. First, we identified common elements involved in the architecture of CAC schemes and provided an architectural model expressing the set $\mathcal{ARC}$ of the possible architectures. Then, we showed how to evaluate different architectures based on security and usability goals. To find the optimal architecture, we formalized a MOOP so to leverage well-known techniques for Pareto optimality. For concreteness, we gave a proof-of-concept application of how the architectural model and the MOOP can be used to assist administrators in the deployment of CAC schemes architectures. We implemented a web dashboard to solve the specific formalization of the optimization problem and perform a "What-if" analysis on the resulting architecture to carefully assess the trade-offs of the security and usability goals. We used the TOSCA OASIS standard and the Cloudify framework to automatize the deployment of the architecture. Finally, we chose a CAC scheme supporting the architecture and provide a fully working prototype with AWS.

Future Directions. While being an example and not the focus of this work, the goals we identified may not be enough to express the requirements of all scenarios. Therefore, we could investigate further scenarios like eBusiness, eBanking and FinTech. Another interesting improvement would be extending our tool to support additional blueprints associated with more CAC schemes. A security evaluation of the 81 architectures could be performed along with the optimization to explicit security assumptions and requirements.

REFERENCES

[1] Assad Abbas and Samee U. Khan. A Review on the State-of-the-Art Privacy-Preserving Approaches in the e-Health Clouds. *IEEE Journal of Biomedical and Health Informatics*, 18(4):1431–1441, July 2014.

[2] Mikhail J. Atallah, Marina Blanton, Nelly Fazio, and Keith B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3):18:1–18:43, January 2009.

[3] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, 05 2007.

[4] 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 , 01 2014.

[5] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):222–233, January 2014.

[6] Marios D. Dikaiakos, Dimitrios Katsaros, Pankaj Mehra, George Pallis, and Athena Vakali. Cloud Computing: Distributed Internet Computing for IT and Scientific Research. *IEEE Internet Computing*, 13(5):10–13, September 2009.

[7] Josep Domingo-Ferrer, Oriol Farràs, Jordi Ribes-González, and David Sánchez. Privacy-preserving cloud computing on sensitive data: A survey of methods, products and challenges. *Computer Communications*, 140-141:38–60, May 2019.

[8] Anna Lisa Ferrara, Georg Fuchsbauer, Bin Liu, and Bogdan Warinschi. Policy Privacy in Cryptographic Access Control. In *2015 IEEE 28th Computer Security Foundations Symposium*, pages 46–60, Verona, July 2015. IEEE.

[9] Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Encryption policies for regulating access to outsourced data. *ACM Transactions on Database Systems (TODS)*, 35:12, 04 2010.

[10] 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 *2016 IEEE Symposium on Security and Privacy (SP)*, pages 819–838, San Jose, CA, May 2016. IEEE.

[11] Valentin Ghita, Sergiu Costea, and Nicolae Tapus. Implementation of cryptographically enforced rbac. *The Scientific Bulletin - University Politehnica of Bucharest*, 79(2):9–3–102, 2017.

[12] 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*, pages 579–591, 07 2008.

[13] 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*, pages 89–98, 01 2006.

[14] Horst W. Hamacher, Christian Roed Pedersen, and Stefan Ruzika. Multiple objective minimum cost flow problems: A review. *European Journal of Operational Research*, 176(3):1404–1422, February 2007.

[15] Felix Horandner, Stephan Krenn, Andrea Migliavacca, Florian Thiemer, and Bernd Zwattendorfer. CREDENTIAL: A Framework for Privacy-Preserving Cloud-Based Data Sharing. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 742–749, Salzburg, Austria, August 2016. IEEE.

[16] 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*, pages 466–481, 04 2002.

[17] Yashpalsinh Jadeja and Kirit Modi. Cloud computing - concepts, architecture and challenges. In *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, pages 877–880, Nagercoil, Tamil Nadu, India, March 2012. IEEE.

[18] Julian Jang-Jaccard. A Practical Client Application Based on Attribute Based Access Control for Untrusted Cloud Storage. In *Computer Science & Information Technology*, pages 01–15. Academy & Industry Research Collaboration Center (AIRCC), January 2018.

[19] Md. Tanzim Khorshed, A.B.M. Shawkat Ali, and Saleh A. Wasimi. A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing. *Future Generation Computer Systems*, 28(6):833–851, June 2012.

[20] Rakesh Kumar and Rinkaj Goyal. On cloud security requirements, threats, vulnerabilities and countermeasures: A survey. *Computer Science Review*, 33:1–48, August 2019.

[21] Thomas Loruenser, Daniel Slamanig, Thomas Langer, and Henrich C. Pohls. PRISMACLOUD Tools: A Cryptographic Toolbox for Increasing Security in Cloud Services. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 733–741, Salzburg, Austria, August 2016. IEEE.

[22] R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, April 2004.

[23] Sascha Muller and Stefan Katzenbeisser. Hiding the policy in cryptographic access control. In *Security and Trust Management*, pages 90–105, 2012.

[24] 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*, pages 195–203, 01 2007.

[25] Praveen Kumar P, Syam Kumar P, and Alphonse P.J.A. Attribute based encryption in cloud computing: A survey, gap analysis, and future directions. *Journal of Network and Computer Applications*, 108:37–52, April 2018.

[26] David W. Pentico. Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2):774–793, January 2007.

[27] R. Perlman. File System Design with Assured Delete. In *Third IEEE International Security in Storage Workshop (SISW'05)*, pages 83–88, San Francisco, CA, USA, 2005. IEEE.

[28] Uthpala Premarathne, Alsharif Abuadbba, Abdulatif Alabdulatif, Ibrahim Khalil, Zahir Tari, Albert Zomaya, and Rajkumar Buyya. Hybrid Cryptographic Access Control for Cloud-Based EHR Systems. *IEEE Cloud Computing*, 3(4):58–64, July 2016.

[29] Saiyu Qi and Yuanqing Zheng. Crypt-DAC: Cryptographically Enforced Dynamic Access Control in the Cloud. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2019.

[30] Gururaj Ramachandra, Mohsin Iftikhar, and Farrukh Aslam Khan. A Comprehensive Survey on Security in Cloud Computing. *Procedia Computer Science*, 110:465–472, 2017.

[31] E. Ramirez, J. Brill, M.K. Ohlhausen, J.D. Wright, and T. McSweeny. Data brokers: A call for transparency and accountability. In *Data brokers: A call for transparency and accountability*, pages 1–101. CreateSpace Independent Publishing Platform, January 2014.

[32] Fatemeh Rezaeibagha and Yi Mu. Distributed clinical data sharing via dynamic access-control policy transformation. *International Journal of Medical Informatics*, 89:25–31, May 2016.

[33] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[34] Pierangela Samarati and Sabrina de Capitani di Vimercati. Access control: Policies, models, and mechanisms. In Riccardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171, pages 137–196. Springer Berlin Heidelberg, 2000.

[35] Ravi Sandhu. Access control: principle and practice. *Advances in Computers*, 46:237 – 286, 10 1998.

[36] Hiroyuki Sato and Somchart Fugkeaw. Design and Implementation of Collaborative Ciphertext-Policy Attribute-Role based Encryption for Data Access Control in Cloud. *Journal of Information Security Research*, 6(3):71–84, September 2015.

[37] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[38] Ashish Singh and Kakali Chatterjee. Cloud security issues and challenges: A survey. *Journal of Network and Computer Applications*, 79:88–115, February 2017.

[39] Yang Tang, Patrick P.C. Lee, John C.S. Lui, and Radia Perlman. Secure Overlay Cloud Storage with Access Control and Assured Deletion. *IEEE Transactions on Dependable and Secure Computing*, 9(6):903–916, November 2012.

[40] Saman Zarandioon, Danfeng Yao, and Vinod Ganapathy. K2c: Cryptographic Cloud Storage with Lazy Revocation and Anonymous Access. In Muttukrishnan Rajarajan, Fred Piper, Haining Wang, and George Kesidis, editors, *Security and Privacy in Communication Networks*, volume 96, pages 59–76. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[41] Lan Zhou, Vijay Varadharajan, and Michael Hitchens. Achieving Secure Role-Based Access Control on Encrypted Data in Cloud Storage. *IEEE Transactions on Information Forensics and Security*, 8(12):1947–1960, December 2013.

A ARCHITECTURES AND GOALS

We report here the complete list (in two tables, Table 4 and 5) of all architecture $arc \in \mathcal{ARC}$ along with the objective functions $g_{\text{reliability}}(arc), ..., g_{\text{CSP Savings}}(arc)$ on the security and usability goals.

Table 4: Architectures and Goals (1 of 2)

Client	Domain On-premise	Cloud	Redundancy	Scalability	Reliability	Maintenance	DoS Resilience	Vendor Lock-in	On-premise Savings	CSP Savings
			+3	+3	+3	+3	+3	−3	+3	−3
			+2	+2	+2	+2	+2	−2	+2	−2
			+1	+1	+1	+1	+1	−1	+1	−1
			+2	+2	+2	+2	+2	−2	+2	−2
			+1	+1	+1	+1	+1	−1	+1	−1
			0	0	0	0	0	0	0	0
			+1	+1	+1	+1	+1	−1	+1	−1
			0	0	0	0	0	0	0	0
			−1	−1	−1	−1	−1	+1	−1	+1
			+2	+2	+2	+2	+2	−2	+2	−2
			+1	+1	+1	+1	+1	−1	+1	−1
			0	0	0	0	0	0	0	0
			+1	+1	+1	+1	+1	−1	+1	−1
			0	0	0	0	0	0	0	0
			−1	−1	−1	−1	−1	+1	−1	+1
			0	0	0	0	0	0	0	0
			−1	−1	−1	−1	−1	+1	−1	+1
			−2	−2	−2	−2	−2	+2	−2	+2
			+1	+1	+1	+1	+1	−1	+1	−1
			0	0	0	0	0	0	0	0
			−1	−1	−1	−1	−1	+1	−1	+1
			0	0	0	0	0	0	0	0
			−1	−1	−1	−1	−1	+1	−1	+1
			−2	−2	−2	−2	−2	+2	−2	+2
			−1	−1	−1	−1	−1	+1	−1	+1
			−2	−2	−2	−2	−2	+2	−2	+2
			−3	−3	−3	−3	−3	+3	−3	+3
			+3	+2	+2	+2	+2	−3	+2	−3
			+2	+1	+1	+1	+1	−2	+1	−2
			+1	0	0	0	0	−1	0	−1
			+2	+1	+1	+1	+1	−2	+1	−2
			+1	0	0	0	0	−1	0	−1
			0	−1	−1	−1	−1	0	−1	0
			+1	0	0	0	0	−1	0	−1
			0	−1	−1	−1	−1	0	−1	0
			−1	−2	−2	−2	−2	+1	−2	+1
			+2	+1	+1	+1	+1	−2	+1	−2

Table 5: Architectures and Goals (2 of 2)

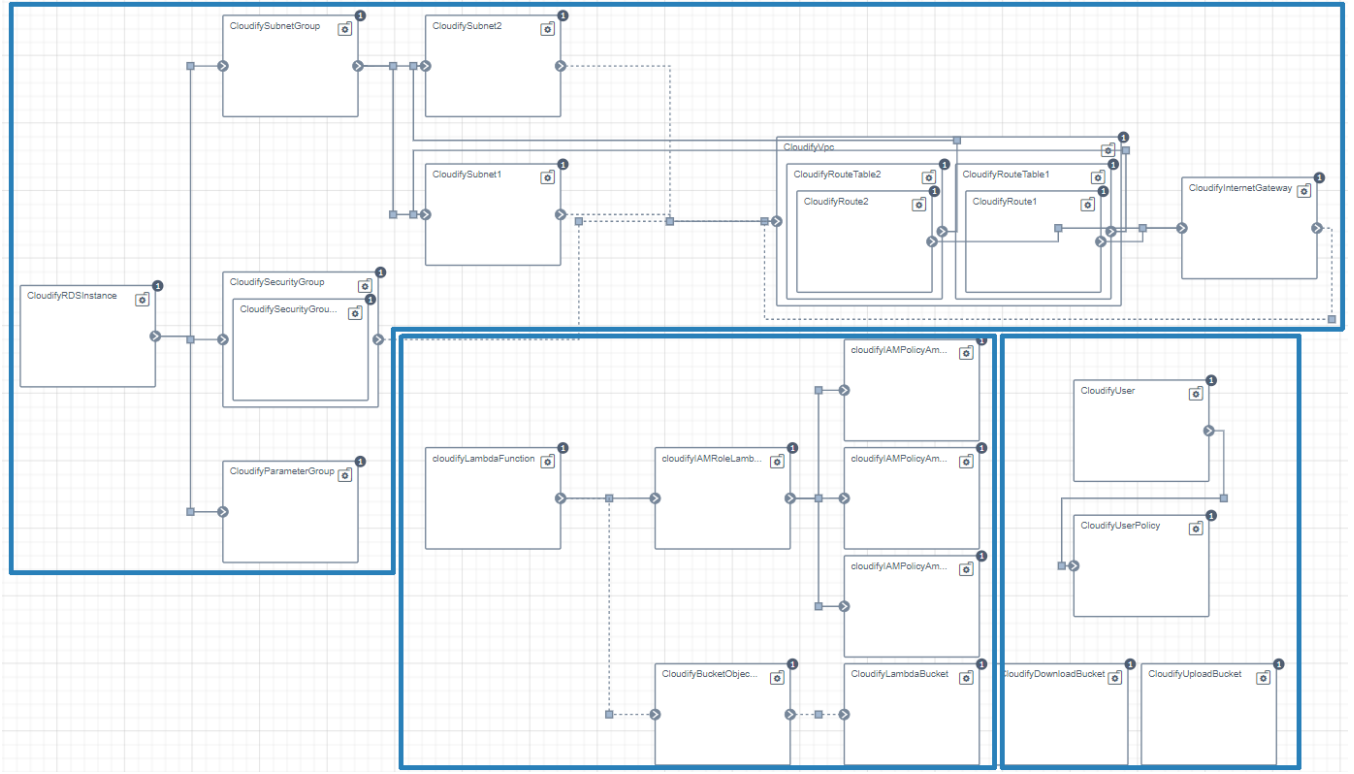| Client	Domain On-premise	Cloud	Redundancy	Scalability	Reliability	Maintenance	DoS Resilience	Vendor Lock-in	On-premise Savings	CSP Savings
			+1	0	0	0	0	−1	0	−1
			0	−1	−1	−1	−1	0	−1	0
			+1	0	0	0	0	−1	0	−1
			0	−1	−1	−1	−1	0	−1	0
			−1	−2	−2	−2	−2	+1	−2	+1
			0	−1	−1	−1	−1	0	−1	0
			−1	−2	−2	−2	−2	+1	−2	+1
			−2	−3	−3	−3	−3	+2	−3	+2
			+1	0	0	0	0	−1	0	−1
			0	−1	−1	−1	−1	0	−1	0
			−1	−2	−2	−2	−2	+1	−2	+1
			0	−1	−1	−1	−1	0	−1	0
			−1	−2	−2	−2	−2	+1	−2	+1
			−2	−3	−3	−3	−3	+2	−3	+2
			−1	−2	−2	−2	−2	+1	−2	+1
			−2	−3	−3	−3	−3	+2	−3	+2
			−3	−4	−4	−4	−4	+3	−4	+3
			+3	+4	+4	+4	+4	−3	+4	−3
			+2	+3	+3	+3	+3	−2	+3	−2
			+1	+2	+2	+2	+2	−1	+2	−1
			+2	+3	+3	+3	+3	−2	+3	−2
			+1	+2	+2	+2	+2	−1	+2	−1
			0	+1	+1	+1	+1	0	+1	0
			+1	+2	+2	+2	+2	−1	+2	−1
			0	+1	+1	+1	+1	0	+1	0
			−1	0	0	0	0	+1	0	+1
			+2	+3	+3	+3	+3	−2	+3	−2
			+1	+2	+2	+2	+2	−1	+2	−1
			0	+1	+1	+1	+1	0	+1	0
			+1	+2	+2	+2	+2	−1	+2	−1
			0	+1	+1	+1	+1	0	+1	0
			−1	0	0	0	0	+1	0	+1
			0	+1	+1	+1	+1	0	+1	0
			−1	0	0	0	0	+1	0	+1
			−2	−1	−1	−1	−1	+2	−1	+2
			+1	+2	+2	+2	+2	−1	+2	−1
			0	+1	+1	+1	+1	0	+1	0
			−1	0	0	0	0	+1	0	+1
			0	+1	+1	+1	+1	0	+1	0
			−1	0	0	0	0	+1	0	+1
			−2	−1	−1	−1	−1	+2	−1	+2
			−1	0	0	0	0	+1	0	+1
			−2	−1	−1	−1	−1	+2	−1	+2
			−3	−2	−2	−2	−2	+3	−2	+3

Figure 4: Cloudify Blueprint Corresponding to $arc^* = \{\langle \textbf{proxy}_u, \textbf{client}_u \rangle, \langle \textbf{RM, CSP} \rangle, \langle \textbf{MS, CSP} \rangle, \langle \textbf{DS, CSP} \rangle\}$

B   AN EXAMPLE OF CLOUDIFY BLUEPRINT

We report here the blueprint fragments composing the Cloudify blueprint for the architectures $arc^* \in \mathcal{ARC}$ that we considered for the eGovernment scenario. Each white rectangle is a node and it represents a cloud service (e.g., security groups, network gateways). Links are relationships between nodes and are used to control the deployment flow. For instance, a "depends_on" relationship from a subnet to a network means that the network has to be deployed first. The blueprint contains three fragments (blue borders). The fragment on top models the MS entity as a Relational Database Service in AWS, while the fragment in the middle models the RM as a Lambda function in AWS. The last fragment on the bottom-right corner models the DS as the S3 service in AWS. The proxy runs in the users' computers and therefore is not part of the blueprint.