Formal Modelling and Automated Trade-Off Analysis of Enforcement Architectures for Cryptographic Access Control in the Cloud

STEFANO BERLATO, University of Genoa and Fondazione Bruno Kessler
ROBERTO CARBONE, Fondazione Bruno Kessler
ADAM J. LEE, University of Pittsburgh
SILVIO RANISE, University of Trento and Fondazione Bruno Kessler

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 these CAC schemes assumes 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 trust assumptions and requirements. For instance, assuming that the proxy runs at the premises of the organization avoids the vendor lock-in effect but may give rise to other security concerns (e.g., malicious insiders attackers).

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 trust assumptions and requirements of a given scenario. In this paper, we propose a methodology to assist administrators in exploring different architectures for the enforcement of CAC schemes in 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 architectures satisfying a heterogeneous set of trust assumptions and requirements arising from the considered scenario to a decidable Multi-Objective Combinatorial Optimization Problem (MOCOP) for which state-of-the-art solvers can be invoked. Finally, we show how we use the capability of solving the MOCOP to build a prototype tool assisting administrators to preliminarily 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.*

Additional Key Words and Phrases: Cryptographic Access Control, Architecture, Optimization

ACM Reference Format:
Stefano Berlato, Roberto Carbone, Adam J. Lee, and Silvio Ranise. 2021. Formal Modelling and Automated Trade-Off Analysis of Enforcement Architectures for Cryptographic Access Control in the Cloud. *ACM Trans. Priv. Sec.* 1, 1 (July 2021), 35 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' addresses: Stefano Berlato, sberlato@fbk.eu, DIBRIS, University of Genoa, Genoa, Italy, Security and Trust Research Unit, Fondazione Bruno Kessler, Trento, Italy; Roberto Carbone, carbone@fbk.eu, Security and Trust Research Unit, Fondazione Bruno Kessler, Trento, Italy; Adam J. Lee, Computer Science Department, University of Pittsburgh, Pittsburgh, United States, adamlee@cs.pitt.edu; Silvio Ranise, silvio.ranise@unitn.it, ranise@fbk.eu, Department of Mathematics, University of Trento, Trento, Italy, Security and Trust Research Unit, Fondazione Bruno Kessler, Trento, Italy.

2471-2566/2021/7-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 CSP itself. Several CAC schemes have been proposed in the literature, each embodying particular features through different cryptographic primitives. Some CAC schemes [17, 22, 45] employ Attribute-Based Encryption (ABE) due to its ability to enforce rich Attribute-Based AC (ABAC) policies. Other schemes combine asymmetric and symmetric cryptography in hybrid cryptosystems [12], employ lazy revocation [46] or express other AC models like Role-Based AC (RBAC) [47]. Others adopt proxy re-encryption [37] or onion encryption [34] 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 concrete use [12]. For instance, ABE applied to AC in the cloud "exists in an academic world and it is often difficult to find a practical use of ABE for a real application" [22]. Since researchers usually focus on high-level features only, little space is left for aspects related to the use of their scheme in a given scenario. An important aspect for the deployment of CAC schemes is the definition of the entities that compose the scheme along with the entities' logical or physical locations (i.e., the definition of the "architecture" of the CAC scheme). However, researchers seldom provide an architecture for their CAC scheme, or this is usually fixed and it cannot adapt to the trust assumptions (e.g., presence of malicious insider or external attackers) and requirements (e.g., enhance architecture scalability or reduce monetary costs) of different scenarios. Indeed, while CAC has been studied in several scenarios like eHealth [1, 9, 19, 30, 33, 37] and eGovernment [19, 27], we note that different scenarios have different trust assumptions and requirements. For instance, the eHealth scenario may demand stricter control over appliances managing medical data, while the eGovernment scenario may require to enhance the scalability and reliability of the architecture. Unfortunately, the lack of study on the relationship between the architectures and the trust assumptions and requirements of different scenarios hampers the adoption of CAC schemes. 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 deployment in a given scenario.

Solution. This paper provides formal modelling and automated trade-off analysis to find the optimal CAC scheme architectures for the trust assumptions and requirements of a given scenario for the enforcement of CAC schemes in the cloud. In detail, our contributions are as follows:

- we discuss centralized vs. decentralized AC enforcements to define under which conditions traditional AC suffices to protect sensitive data from both external attackers and partially trusted CSPs, instead of relying on more complicated (and likely computationally demanding [12]) decentralized solutions;
- we provide a formal architectural model to capture elements—namely resources, domains and entities—commonly involved in the architectures of CAC schemes. Through constraint satisfaction, the architectural model synthesizes the set of architectures preserving the expected confidentiality, integrity and availability properties of the involved resources for CAC schemes. We call these the "candidate" architectures. Then, to validate the generality of the model, 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 according to risk levels (i.e., concerning confidentiality, integrity and availability of sensitive data) derived from the trust assumptions and security and usability goals (e.g., scalability and reliability) that may be desirable in specific deployment scenarios. Then, we formalize a Multi-Objective Combinatorial Optimization

Problem (MOCOP) [24] over the set of candidate architectures to perform a trade-off analysis aiming to strike the best possible balance between security and objectives of other natures. We highlight that, when considering only risk levels and not goals, the trade-off analysis reduces to a risk assessment, i.e., to the problem of finding the most secure architectures for a given scenario. Instead, when considering only goals like scalability and reliability, the trade-off analysis reduces to the problem of finding the most performant architectures for a given scenario. Finally, by reusing, off-the-shelf, well-known techniques for solving the MOCOP by analyzing (apparently equivalent) optimal solutions, also known as "Pareto Optimal" solutions, we provide automated support to the difficult and time-consuming process of selecting the most suitable architecture for a given scenario;

- we give a proof-of-concept application of how the architectural model and the trade-off analysis can be used to assist administrators in the deployment of CAC schemes architectures. We develop a web dashboard[1] implementing two different algorithms to solve the MOCOP, i.e., `Best` from [14] and an ad-hoc algorithm. Through a "What-if" analysis, the former allows evaluating architectures when the requirements of the scenario over the goals (e.g., enhance redundancy) and the risk levels (e.g., preserve the availability of data) are unknown a-priori, while the latter exploits the knowledge of such requirements to find the optimal CAC scheme architectures for the underlying scenario. Furthermore, we study the computational complexity of the two algorithms and measure their efficiency in the best, worst and average cases. To ensure cloud portability, interoperability and automatic deployment, we rely on the TOSCA[2] (Topology and Orchestration Specification for Cloud Applications) OASIS standard to automatically generate a deployable specification of the architectures. Finally, we implement a CAC scheme (i.e., the scheme proposed in [12]) into a fully working prototype and deploy it with Amazon Web Services (AWS).

The contributions of this paper are built on top of the methodology we previously presented in [3]. In [3], we provided an architectural model to formally express the set of the possible architectures for cloud-based CAC schemes. Then, we defined how to evaluate these architectures with security and usability goals. Finally, we proposed a single-objective optimisation problem to identify the best architecture for a given scenario. This paper broadens the scope of [3] by first defining under which conditions the use of (computationally demanding) CAC schemes is really necessary (Section 4). Then, we propose a meticulous risk assessment obtained by relaxing (some of) the trust assumptions (implicit in [3]) to evaluate the risk levels of CAC schemes architectures against a heterogeneous set of attackers (e.g., Man-in-the-Middle and malicious insiders). Indeed, when considering only the goals in [3], organisations may end up choosing well-performing architectures which, however, mine the security of the sensitive data due to some not-fulfilled trust assumptions. To solve this issue, we make such trust assumptions explicit to allow organisations to make more informed decisions by transparently evaluating the risk exposure of CAC schemes architectures and investigating the interplay with the other goals (Section 6). Moreover, as shown in Table 1, we propose a two-fold generalization of the original approach in [3] by (*i*) including the risk levels in the optimisation problem (Section 7.4) and (*ii*) developing the capability to handle scenarios in which requirements are unknown a-priori; this ability empowers organisations to reuse off-the-shelf any approach available in the literature to solve MOCOPs by identifying Pareto Optimal solutions (Section 7.5). Finally, this paper integrates two different algorithms for solving the optimisation problem and studies their computational complexity, describes their implementation in a web dashboard and conducts an efficiency and performance evaluation (Section 8.1). As a final remark, we highlight that our contributions are independent of the underlying CAC scheme and AC policy model.

[1] see https://stfbk.github.io/complementary/TOPS2020_2

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

Table 1. Graphical Representation of the Two-Fold Generalization of the Optimisation Problem

		Requirements Known	
		A-Priori	A-Posteriori
Objective Functions	Goals	this work	[3]
	Protection Levels	this work	this work

The paper is structured as follows. In Section 2 we introduce the background. In Section 3, we illustrate two important scenarios often considered in cloud-relevant literature, namely eHealth and eGovernment, while in Section 4 we discuss centralized vs. decentralized AC enforcements. In Section 5, we introduce our architectural model based on common elements of CAC schemes. We discuss trust assumptions and perform a risk assessment on the architectures in Section 6. We show how to evaluate architectures based on risk levels and security and usability goals and formalize the MOCOP along with two algorithms to solve it in Section 7. We present the web dashboard, the complexity analysis and our proof-of-concept deployment with AWS in Section 8. In Section 9, we discuss related work and conclude the paper with final remarks and future work in Section 10.

2 BACKGROUND

Below, we describe AC, RBAC and the high-level functioning of a cryptographic RBAC scheme.

2.1 Access Control

Samarati and De Capitani di Vimercati [39] 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 [40] and ABAC [20] are two models) giving the semantics for 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 definition of the entities that compose the scheme along with the entities' logical or physical locations (i.e., the architecture) is part of the enforcement level.

We highlight that the 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 a *permission* p if $\exists r : ((u, r) \in UR) \wedge ((r, p) \in PA)$. 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 [5, 11]. A partially trusted domain is a domain controlled by a third-party (e.g., an external organization or a CSP) 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, as traditionally assumed in the literature of cloud computing [6]. Indeed, a report by the U.S. Federal Trade Commission [36] 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 [12] for enforcing cryptographic RBAC policies, although our findings can be generalized for other CAC schemes (e.g., [33, 37, 45–47]). 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 $(\mathbf{k}_u^{\mathrm{s}}, \mathbf{k}_u^{\mathrm{p}})$ and $(\mathbf{k}_r^{\mathrm{s}}, \mathbf{k}_r^{\mathrm{p}})$, respectively. Each file is encrypted with a different symmetric key $\mathbf{k}^{\mathrm{sym}}$. To assign a *user* to a *role*, the *role*'s secret key $\mathbf{k}_r^{\mathrm{s}}$ is encrypted with $\mathbf{k}_u^{\mathrm{p}}$, resulting in $\{\mathbf{k}_r^{\mathrm{s}}\}_{\mathbf{k}_u^{\mathrm{p}}}$. To give `read` *permission* to a *role* over some data, the symmetric key $\mathbf{k}^{\mathrm{sym}}$ related to the data is encrypted with $\mathbf{k}_r^{\mathrm{p}}$, resulting in $\{\mathbf{k}^{\mathrm{sym}}\}_{\mathbf{k}_r^{\mathrm{p}}}$. The use of both secret-public and symmetric cryptography is usually called "Hybrid Encryption" [12]. 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. *Users* store their private key in secure personal devices (e.g., laptops provided with an antivirus) whose access is protected through passwords or similar authentication techniques, while both encrypted data and metadata are stored in the cloud or in a secure area within the organization. To read a file, a *user* u performs the following actions through a software entity usually called proxy:

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

To write on a file, a *user* u performs the same operations to obtain the symmetric key $\mathbf{k}^{\mathrm{sym}}$ which is used to encrypt the new file. Finally, an entity usually called Reference Monitor (RM) checks whether u has `write` *permission* before accepting the new file and storing it in the cloud.

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 the presence of a partially trusted Cloud Service Provider (CSP). Each scenario is characterized by specific trust assumptions (e.g., on the employees of the organization) and requirements (e.g., simplify maintenance or enhance reliability) that are affected by different CAC scheme architectures. For instance, the architecture of the CAC scheme presented in [12] and reported in Section 2.2 assumes the data, metadata and RM to stay in the cloud, 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 CSP. Hosting the RM at the premises of the organization may partially relieve these issues but could create other concerns (e.g., malicious insiders tampering with the RM). 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 eHealth and eGovernment. We discuss possible trust assumptions and requirements and highlight the importance of carefully analysing architectural trade-offs when deploying a CAC scheme.

3.1 eHealth Scenario

The problem of storing medical data in the cloud has been widely studied in the literature [1] by many researchers [9, 19, 27, 30, 33, 37, 41, 46], along with the eHealth scenario trust assumptions and requirements. For instance, Hörandner et al. [19] 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. [9] 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 hospitalized in a clinic specialized for treating such a type of disorders employing well-paid doctors. 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 as the name of the file); the CSP may then infer that a specific person is a customer of the clinic. 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. Below, we summarize some of the most important trust assumptions (A) and requirements (R) of the eHealth (eH) scenario:

- eHA1 - there is a low probability of disgruntled doctors;
- eHA2 - CSP may be curious about medical data and metadata;

- eHR1 - need to hide metadata to avoid information leaking;
- eHR2 - prioritize redundancy to avoid medical data loss.

It should be clear how difficult it is to select the most suitable CAC scheme architectures for the eHealth scenario, as this means finding the architectures that maximize the satisfaction of all requirements over both goals and risk levels derived from the given trust assumptions.

3.2 eGovernment Scenario

The eGovernment scenario is getting more attention [19, 27] as the Public Administrations (PAs) in different countries (e.g., Italy[3] and Spain[4]) start a digitalization process to blend their infrastructure with the cloud.[5]

Based on technologies that include mobile and web applications together with electronic identity services, PAs can develop a portfolio of public services. Suppose a PA wants to allow citizens to access government-issued personal documents (e.g., tax certificates) from anywhere. A European citizen may use eIDAS[6] to authenticate to 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 [19]. We summarize some of the most important trust assumptions and requirements of the eGovernment (eG) scenario:

- eGA1 - the presence of disgruntled employees is possible;
- eGA2 - communication channels may not always be secure;

[3] https://www.agid.gov.it/it/infrastrutture/ [4] https://joinup.ec.europa.eu/collection/egovernment/news/spanish-government-approv
[5] https://joinup.ec.europa.eu/ [6] https://www.eid.as/home

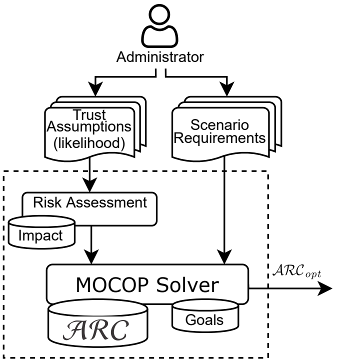


Fig. 1. Tool-Supported Methodology Flow

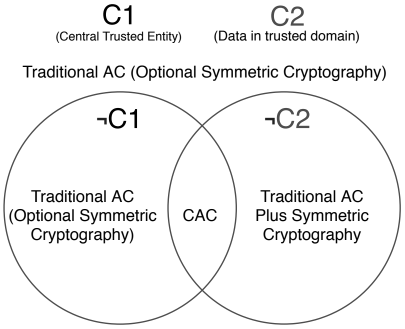

C1
(Central Trusted Entity)
C2
(Data in trusted domain)
Traditional AC (Optional Symmetric Cryptography)
¬C1
¬C2
Traditional AC (Optional Symmetric Cryptography)
CAC
Traditional AC Plus Symmetric Cryptography

Fig. 2. Conditions and Inherent AC Enforcement

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

As for the eHealth scenario, it is not trivial how to find the architectures 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 set of CAC scheme architectures that strike the best possible trade-off considering the trust assumptions and requirements of a scenario. We argue that there is no single CAC scheme architecture that fits all scenarios. Instead, there is a need to carefully evaluate different architectures and find the ones that optimize both the goals and the risk levels derived from the trust assumptions of each scenario. We do this by developing a tool-supported methodology that assists administrators in finding the best possible trade-off. The high-level flow of our approach is reported in Figure 1, where $\mathcal{ARC}$ is the set of all candidate architectures for CAC schemes (the formal definition of candidate architecture is given in Section 5): the administrator provides as input the trust assumptions and the requirements of a scenario. Given the trust assumptions, we assign a likelihood to a set of possible attackers (e.g., malicious insider) and, based on a precomputed impact, perform a risk assessment (Section 6) to assign risk levels to each architecture. Besides, we consider the requirements on the risk levels and the security and usability goals and discuss how much each architecture fulfils these requirements (Section 7). Finally, we reduce the problem of finding the set of architectures $\mathcal{ARC}_{opt}$ that maximize the satisfaction of the requirements to a MOCOP [24] (Section 7.4).

4 CENTRALIZED VS. DECENTRALIZED AC

Before discussing the architectures of CAC schemes, we briefly analyse the relationship between the mechanism chosen to enforce AC policies and the related architecture. We argue that, under certain conditions, traditional (i.e., centralized) AC may suffice to protect sensitive data from both external attackers and partially trusted CSPs, instead of relying on more complicated (and likely computationally demanding [12]) CAC schemes.

Below, we discuss architectural conditions that can make one class of AC enforcements more suitable than another for a specific architecture. As introduced in Section 2.1, there are two main classes of enforcements for AC. In the first class (*centralized*), sensitive data and a central entity mediating users' accesses are both placed in a fully trusted domain (e.g., at the premises of the organization). In the second class (*decentralized*), sensitive data and the entity mediating users' accesses are deployed in a partially trusted domain.

From these definitions, we derive that there are two architectural conditions at the base of the centralized class of AC enforcements (the negation of these conditions leads to the alternative decentralized class):

- C1: there exists a central entity deployed in a fully trusted domain that mediates every access to sensitive data;
- C2: sensitive data are stored in a fully trusted domain.

As shown in Figure 2, there are four possible cases to consider:

- when both C1 and C2 are true (i.e., C1∧C2), the central trusted entity can easily enforce AC policies over securely stored data by following traditional specifications (e.g., XACML[7]). This is the common case in which an organization internally hosts and manages accesses to sensitive data (e.g., private cloud). Even though not strictly required, symmetric cryptography can enhance the security of sensitive data;
- when C1 is true and C2 is false (i.e., C1∧¬C2), data are stored by a third-party CSP and users' access is mediated by a trusted entity deployed within the organization. We highlight that assuming the presence of a central trusted entity obviates the need for any cryptography beyond authenticated symmetric key encryption [12]. Data (e.g., files) can be encrypted with symmetric keys (e.g., one per file) that are securely stored by the central entity. For every (authenticated) user's access request, the central entity autonomously decides whether to accept (i.e., by decrypting the file and returning it to the user) or reject the user's request based on the AC policy. The same procedure applies to `write` requests. The partially trusted domain (e.g., CSP) cannot read data since the symmetric keys are managed by the central entity. The revocation of users is trivial since it implies modifying the AC policy without making any change to the encrypted data or symmetric keys.
  As an alternative, the central entity can just return the symmetric keys to authorized users which then freely interact with the partially trusted domain to download files. This approach offloads cryptographic operations from the central entity to the users. To read a file, users first request the symmetric key to the central entity and then download the encrypted file, performing the decryption locally. Users follow the same procedure also when writing to a file, i.e., users send the new file to the central entity that either accepts or rejects the new file. The revocation is still trivial since authorized users, when writing, can perform the (lazy) re-encryption of files with a new symmetric key.
  In both alternatives, AC can be enforced with a traditional specification, while symmetric cryptography suffices for preserving the confidentiality of outsourced data;
- when C1 is false and C2 is true (i.e., ¬C1∧C2), data are stored in a trusted domain. Since a central trusted entity does not exist, users (usually) access data directly. Even if unusual, users could also access data through a third-party CSP (e.g., hybrid cloud). In this case, the organization would outsource the management of the AC policy to the CSP, which authenticates users and distributes access tokens (e.g., OAuth2[8]) through which users access data. Again, even though not strictly required, symmetric cryptography can enhance the security of sensitive data;
- when both C1 and C2 are false (i.e., ¬C1∧¬C2), there is no central trusted entity and data are stored in a partially trusted domain (e.g., public cloud). Therefore, the only way to enforce

[7] http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html [8] https://oauth.net/2/

the AC policy and preserve data confidentiality is the use of decentralized AC mechanisms such as CAC schemes (see Section 2).

Centralized AC enforcement (plus possibly symmetric cryptography) can be used in almost all possible cases (3 out of 4) instead of computationally expensive CAC schemes. However, we note that this is a general consideration and it is conditional to the presence of factors influencing a specific scenario. In particular, we identify three categories of factors that may drive an organization toward the use of CAC schemes instead or on top of centralized AC enforcements:

- *security-related* factors: while the use of CAC schemes worsens the performance of the whole system, having multiple AC layers is an enhancement from the security point of view;
- *business-related* factors: corporate rules and digitalization strategies (e.g., moving the internal infrastructure to a public cloud) may constrain or guide the choices of organizations;
- *privacy-related* factors: compliance with privacy regulations (e.g., the GDPR[9]) may foster the adoption of CAC schemes. For instance, SAFE[10], a cloud solution for ski area managers, uses a CAC scheme to handle personal data so to be unable to read the data. In this way, the company avoids being affected by privacy regulations while demonstrating total transparency to its customers.

Depending on the scenario, CAC may be in any case the most suitable solution, even though not the most efficient one. Therefore, in the next sections, we identify the set of the candidate architectures of CAC schemes and show how to evaluate and optimize the satisfaction of all requirements of a specific scenario over both goals and risk levels.

5 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 containing the basic building blocks of our architectural model, namely (cryptographic) Resources, Domains and Entities. We also consider the set Properties containing 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 (*Preserves*) the security properties of resources, while entities use (*Uses*) and host (*Hosts*) resources and inherit (*Inherits*) security properties required (*Requires*) by resources. Figure 3a summarizes these considerations 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 combine to specify architectures for CAC schemes. 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 3 summarizes the concepts expressed in this section.

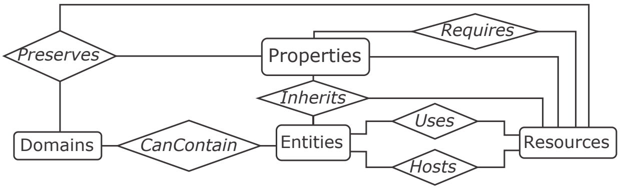
5.1 Cryptographic Resources and Properties

The set Resources contains (cryptographic) resources of the following types (see the column Resource in Table 3b):

- *encrypted data*: by definition, the architecture of a CAC scheme involves data (e.g., a file encrypted with $\mathbf{k}^{\mathbf{sym}}$, 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 user u ($\mathbf{k}^{\mathbf{s}}_{u}, \mathbf{k}^{\mathbf{p}}_{u}$) and role r ($\mathbf{k}^{\mathbf{s}}_{r}, \mathbf{k}^{\mathbf{p}}_{r}$) and the symmetric key for a file $\mathbf{k}^{\mathbf{sym}}$, as introduced in Section 2.2);

[9] https://gdpr.eu/

[10] https://motorialab.com/en/safe/



(a) Elements and Relationships

Resource	Property		
	C	I	A
encrypted data	✗	✓	✓
secret keys	✓	✓	✓
metadata	✗/✓	✓	✓

(b) *Requires*(*res*, *pro*)

Domain	Property		
	C	I	A
client$_u$	★$_u$	★$_u$	★$_u$
on-premise	★	★	★
CSP	-	★	★

(c) *Preserves*(*dom*, *pro*, *res*)

Entity	Resources
🖥 proxy$_u$ proxy	secret keys$_u$ secret keys
🛡 RM	-
🛢 MS	metadata
📁 DS	encrypted data

(d) *Hosts*(*ent*, *res*)

Entity	Resources
🖥 proxy$_u$ proxy	metadata$_u$, encrypted data$_u$ metadata, encrypted data
🛡 RM	metadata, encrypted data
🛢 MS	-
📁 DS	-

(e) *Uses*(*ent*, *res*)

Entity	Property		
	C	I	A
🖥 proxy$_u$ proxy	secret keys$_u$, metadata$_u$[a] secret keys, metadata[a]	★$_u$ ★	secret keys$_u$ secret keys
🛡 RM	metadata[a]	metadata, encrypted data	-
🛢 MS	metadata[a]	metadata	metadata
📁 DS	-	encrypted data	encrypted data

[a] If *Requires*(metadata, C);

(f) *Inherits*(*ent*, *pro*, *res*)

Domain	Entities
client$_u$	🖥[b]
on-premise	🖥 📁 🛡 🛢
CSP	📁 🛡[c] 🛢[c]

[b] proxy$_u$;

[c] If **not** *Requires*(metadata, C);

(g) *CanContain*(*dom*, *ent*)

$$Inherits(ent, pro, res) \text{ iff } \left(\begin{array}{l} (Hosts(ent,\ res) \land Requires(res,\ pro)) \\ \lor (Uses(ent,\ res) \land (Requires(res, C) \lor Requires(res, I))) \end{array} \right) \tag{1}$$

$$\begin{array}{l} CanContain(dom, ent) \text{ iff} \\ \forall pro,\ res.\ \exists res' \left(\begin{array}{l} (Inherits(ent,\ pro,\ res) \implies \\ \quad Preserves(dom,\ pro,\ res')) \land res \sqsubseteq res' \end{array} \right) \end{array} \tag{2}$$

(h) Sets

Fig. 3. Summary of Basic Notions of CAC Schemes

- *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 3b defines the relation *Requires*, i.e., it identifies each CIA property *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 $\langle res, pro\rangle \notin Requires$ when it contains the symbol ✗.

We assume perfect encryption over data (i.e., the confidentiality of 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. 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 its internal hierarchy [47]. 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. However, not all CAC schemes expect to encrypt metadata, and some entities may need to access plain-text metadata anyway. Thus, we consider as optional the possibility to have sensitive metadata.

5.2 Domains

Following [12] and [46], 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 3c):

- $client_u$: is the domain in which the user u operates, and it is defined as the set of user's u personal devices (e.g., laptop and smartphone). As assumed in Section 2.2, personal devices are not shared among users and 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 [7].

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 3c 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, $Preserves(\textit{on-premise}, C, res)$, $Preserves(\textit{on-premise}, I, res)$ as well as $Preserves(\textit{on-premise}, A, res)$. 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 $(CSP, C, res) \notin Preserves$ 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., $(\mathbf{k}^s_u, \mathbf{k}^p_u)$) and not the whole set of secret keys (e.g., another user's u' keys, i.e., $(\mathbf{k}^s_{u'}, \mathbf{k}^p_{u'})$). 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.

5.3 Entities and Relationships with Resources

The set Entities contains elements that actively perform tasks in CAC schemes (see the columns Entity in Table 3d and Table 3e):[11]

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

- *proxy*: Domingo-Ferrer et al. [9] argued that the architectures of CAC schemes usually involve a local proxy to interface users with encrypted data. The proxy encrypts the data before uploading them to the cloud and decrypts them before showing them to the user.
- *reference monitor* (RM): Garrison et al. [12] discussed the presence of a 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);
- *metadata storage* (MS): this entity is the storage (e.g., a database) containing the metadata.
- *data storage* (DS): this entity is the storage (e.g., a simple storage service) containing the data;

To accomplish its tasks, an entity must be located in at least one domain (e.g., a software needs to run on a machine). Furthermore, an entity may host and use resources (e.g., a proxy using a secret cryptographic key to decrypt an encrypted file). Table 3d and Table 3e 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 can retrieve metadata and encrypted data 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 and encrypted data. Formally, we specify these as $Hosts(proxy_u, secret\ keys_u)$, $Uses(proxy_u, metadata_u)$, $Uses(proxy_u, encrypted\ data_u)$ and $Hosts(proxy, secret\ keys)$, $Uses(proxy, metadata)$, $Uses(proxy, encrypted\ data)$. Finally, the RM uses both metadata and encrypted data to verify the compliance with the AC policy of the users' actions; the MS and the DS store metadata and encrypted data, respectively. Formally, this is written as $Uses(RM, metadata)$, $Uses(RM, encrypted\ data)$, $Hosts(MS, metadata)$ and $Hosts(DS, encrypted\ data)$.

5.4 Putting Things Together

By recalling the summary in Figure 3, we are now ready to define the notion of a CAC scheme candidate 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 Figure 3h, 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 either the confidentiality or the integrity property. Indeed, while it is possible to affect the confidentiality (e.g., through information leakage) and integrity (e.g., by tampering with a file before uploading it to the cloud) of a (used) resource, the availability of the original (hosted) resource cannot be compromised. Table 3f is extensionally equivalent to (1) in in Figure 3h.

According to (2) in Figure 3h, 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 is such that 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 and encrypted data$_u$ $\sqsubseteq$ encrypted data as encrypted data$_u$ is a portion of encrypted data. Table 3g is extensionally equivalent to (2) in Figure 3h and can be interpreted as explained below.

If we consider multiple instances of the proxy for each user u (i.e., proxy_u), each instance proxy_u would host a portion of the secret keys (i.e., secret keys_u) and access a portion of the metadata (i.e., metadata_u) and encrypted data (i.e., encrypted data_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. The CSP can contain the RM and MS only if the organization deems metadata not to be sensitive. Otherwise, the RM and MS inherit the confidentiality property of the metadata and therefore cannot stay in the CSP domain. As a final note, since the RM checks users' actions against the AC policy, we assume that the RM cannot run in the users' computer, i.e., we assume that $\langle \text{client}_u, \text{RM} \rangle \notin \mathit{CanContain}$.

As we can see from Table 3g, 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 especially for entities hosting resources (i.e., proxy, MS, DS). For instance, important encrypted files (e.g., with a sensitive name) can be hosted in a DS at the premises of the organisation, while other files can stay in a DS managed by the CSP (e.g., [37]). 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 encrypted data (e.g., [45]). 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., [47]). In these hybrid architectures, 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 satisfy 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 candidate architectures for cloud-based CAC schemes (see Table 2 for a selection and complementary material[12] 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 some CAC schemes [4, 10, 16, 17, 29, 46]. 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 candidate architectures arc as follows:

$$\begin{aligned}\mathcal{ARC} = \{arc \subseteq (\text{Entities} \times \text{Domains}) | \; (\; \forall \langle dom, ent \rangle \in arc : \mathit{CanContain}(dom, \; ent) \;) \wedge \\ (\; \forall ent \in \text{Entities} \setminus \{RM\} \; \exists dom \in \text{Domains} : \langle dom, ent \rangle \in arc \;)\};\end{aligned} \tag{3}$$

5.5 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. [12] designed a CAC scheme for a dynamic RBAC policy with a focus on performance. The architecture comprehends the same three domains that we presented. A proxy for each

[12] https://stfbk.github.io/complementary/TOPS2020_2

Table 2. Considered CAC Scheme Architectures

CAC	Architecture			
Scheme	client$_u$	on-premise	CSP	*arc*
[12]	🖥️		🛢️ 🛡️ 📁	⟨proxy$_u$, client$_u$⟩, ⟨MS, CSP⟩, ⟨RM, CSP⟩, ⟨DS, CSP⟩
[46]	🖥️		🛢️ 📁	⟨proxy$_u$, client$_u$⟩, ⟨MS, CSP⟩, ⟨DS, CSP⟩
[47]	🖥️	🛢️	🛢️ 📁	⟨proxy$_u$, client$_u$⟩, ⟨MS, on-premise⟩, ⟨MS, CSP⟩, ⟨DS, CSP⟩
[45]	🖥️	🖥️ 🛢️ 🛡️	🛢️ 📁	⟨proxy$_u$, client$_u$⟩, ⟨proxy, on-premise⟩, ⟨MS, on-premise⟩, ⟨RM, on-premise⟩, ⟨MS, CSP⟩, ⟨DS, CSP⟩
[33]		🖥️ 🛢️	📁	⟨proxy, on-premise⟩, ⟨DS, on-premise⟩, ⟨DS, CSP⟩
[37]	🖥️	🛢️ 🛡️ 📁	🛢️ 📁	⟨proxy$_u$, client$_u$⟩, ⟨MS, on-premise⟩, ⟨RM, on-premise⟩, ⟨DS, on-premise⟩, ⟨MS, CSP⟩, ⟨DS, CSP⟩

user contains the user's secret keys. The metadata are in the MS and the encrypted files in the DS entity. Both of these entities, together with the RM checking digital signatures, stay in the CSP.

In [46], the authors discussed the same three domains that we presented. The architecture expects a proxy (called "Key-store") for each user containing the user's secret keys. Non-sensitive metadata (i.e., hierarchies and public parameters) are kept in the "Meta-data Directory" (i.e., the MS) in the CSP domain. The "Data Store" (i.e., the DS) stores encrypted data in the CSP domain. As in [10], the authors proposed a CAC scheme without the RM, relying on users to validate `write` operations.

In [47], 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 [45], the authors proposed a CAC scheme along with a prototype named "FADE". Users interact with a proxy that can be deployed 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, storing sensitive metadata (e.g., cryptographic parameters) through threshold secret sharing [42]. Besides, the key managers perform blind decryption on cryptographic keys [32] and allow users to upload and download files. Thus, the key managers act both as MS and RM. Each (encrypted) file is associated with an AC policy and it is stored by the DS in the CSP domain.

Premarathne et al. [33] 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 (i.e., the proxy) is responsible for extracting users' data 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., MS). 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 the proxy's actions are assumed to be legitimate.

In [37], the authors propose a CAC scheme based on a hybrid architecture. A private DS (i.e., in the on-premise domain) stores confidential patients' data (e.g., chronic diseases) and it can be accessed by authorized personnel only. A public DS (i.e., in the 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 CSP has part of the metadata needed by the CAC scheme. Finally, each user (e.g., doctors and nurses) is given a secret key (i.e., proxy$_u$).

6 RISK ASSESSMENT

In Section 2.2, we formulated four trust assumptions (TA) that we report below for ease of reference:

- TA1 - all communications occur through pairwise authenticated and private channels;
- TA2 - users' devices are secure and protected through passwords or similar mechanisms;
- TA3 - the on-premise domain is a secure area within the organization;
- TA4 - the CSP is honest but curious.

However, we note that these trust assumptions may not hold in every scenario. Therefore, to further enhance the general applicability of our architectural model, we present a risk assessment to help administrators in minimizing the level of risk associated with the architectures when considering a set of attackers (e.g., malicious insiders) aiming at compromising the sensitive data.

We follow a common matrix-based approach to risk evaluation (e.g., see [15]), where the risk of an adverse event (e.g., a malicious insider leaking sensitive data) is decomposed in two dimensions, i.e., likelihood and impact, measured with four different values, i.e., N(egligible), L(ow), M(edium), and H(igh). For the likelihood, N and H mean that the probability that an adverse event happens is, for all practical purposes, zero and one, respectively, whereas L and M correspond to intermediate values of the probability. For the impact, N and H mean that the negative consequences of an adverse event happening are negligible and catastrophic, respectively, whereas L and M correspond to intermediate values. The matrix in Figure 4a shows how the levels of likelihood and impact combine to yield a risk level on the same scale with increasing values from N to H.

The impact of an adverse event can be determined by analysing the capabilities of an attacker on the assets of a system. For instance, a virus in a user's device will have access to that user's data only, while a malicious insider with access to a server may compromise more data. Differently, the likelihood crucially depends on the particular scenario in which the system operates. Indeed, the likelihood is related to the probability that an exploitable vulnerability exists and that it will be exploited (e.g., the probability of a bug in a software that is exploited by an attacker). Vulnerabilities due to poor implementation or configuration choices (e.g., using an untested third-party library or not limiting the physical and logical access to servers in an organization) can be more or less difficult to exploit because of mitigations put in place (e.g., a sandbox to constrain the communication between business and personal software or a more restrictive AC policy within the organization). Therefore, while we can precompute the impact, the likelihood depends on the trust assumptions of a specific scenario and must be evaluated by the organization on a case by case basis.

Below, we first define the impact on sensitive data by considering the violation of the confidentiality, integrity and availability of the resources of CAC schemes. Note that we tune our risk assessment at the design level, i.e., considering the abstract categories of resources we identified. Indeed, a more detailed analysis would overly focus on a particular instantiation of the resources for a specific CAC scheme, while we want our findings to apply to our general architectural model. Then, we map resources to entities, discuss possible targets and define the related impact. We characterize attackers and assess the exposure of architectures depending on the variable location of the resources, i.e., in which domains the resources are placed. This allows assigning an impact (high, medium, low or negligible) to the CIA of sensitive data for each combination of architecture-attacker. Figure 4 summarizes the concepts expressed in this section.

6.1 Resources and Impact on Sensitive Data

As a metric, we define in Table 4b the impact of the violation of the CIA properties on each resource to be proportional to the number of files that are affected, i.e., the higher the number of affected files, the higher the impact. For instance, the leakage of a single symmetric key may disclose the

		Likelihood			
		N	L	M	H
Impact	N	N	N	N	N
	L	N	L	L	L
	M	N	L	M	M
	H	N	L	M	H

(a) Risk Table

Resource	Compromised		
	C	I	A
Encrypted Data	N	L	L
Secret Keys	H	N	N
Metadata	N	H	N

(b) Resources and Impact

Resources	Targets
Encrypted Data	🖥$_{ci}$, 📁, 👮$_{ci}$, 📁 $\leftrightarrow_{ci}$ 🖥, 📁 $\leftrightarrow_{ci}$ 👮
Secret Keys	🖥
Metadata	🖥$_{ci}$, 🛢, 👮$_{ci}$, 🛢 $\leftrightarrow_{ci}$ 🖥, 🛢 $\leftrightarrow_{ci}$ 👮

(c) Resources and Targets

Targets	Impact		
	C	I	A
🖥	H	H	N
🛢	N	H	N
👮	N	H	N
📁	N	L	L
📁 ↔ 🖥	N	L	N
🛢 ↔ 🖥	N	H	N
🛢 ↔ 👮	N	H	N
📁 ↔ 👮	N	L	N

(d) Targets and Impact

Architecture			MitD			Malicious Insider						MitM								
client$_u$	on-premise	CSP	Client$_u$			on-premise			CSP			⟨client$_u$, on-premise⟩			⟨client$_u$, CSP⟩			⟨on-premise, CSP⟩		
			C	I	A	C	I	A	C	I	A	C	I	A	C	I	A	C	I	A
🖥		🛢 👮 📁	H	H	N	N	N	N	N	H	L	N	N	N	N	H	N	N	N	N
🖥	🛢 👮	📁	H	H	N	N	H	N	N	L	L	N	H	N	N	L	N	N	L	N
🖥	👮 📁	🛢	H	H	N	N	H	L	N	H	N	N	L	N	N	H	N	N	H	N
🖥	🖥 🛢 👮	📁	H	H	N	H	H	N	N	L	L	N	H	N	N	L	N	N	L	N
	🖥 📁	🛢 👮 📁	N	N	N	H	H	L	N	H	L	N	N	N	N	N	N	N	H	N

(e) (Excerpt of) Architectures and Attackers' Impact

Fig. 4. Summary of Risk Assessment

content of a single file, whereas the leakage of a user's secret key may allow an attacker to access all files the user has permission over.

- *Encrypted Data*: Any violation of encrypted data has a direct impact in terms of integrity and availability of sensitive data, but not on confidentiality. Indeed, as assumed in Section 5.1, the confidentiality of the encrypted data cannot be compromised by attacking the available cryptographic primitives. Finally, we note that the violation of a single encrypted file affects that file only (negligible impact for confidentiality and low for integrity and availability);
- *Secret Keys*: compromising the integrity or the availability of secret keys affect the availability of sensitive data for single users only. For instance, if a user's secret key gets tampered with, then that user cannot use his key to access sensitive data anymore. However, as other users' keys can still be used, this does not have any impact on sensitive data. Instead, the violation of the confidentiality of a secret key is more severe, since the secret key can potentially be used to access (i.e., decrypt) many files, as explained in Section 2 (high impact for confidentiality and negligible for integrity and availability);
- *Metadata*: while mostly depending on the specific CAC scheme, metadata usually contain information like the AC policy, digital signatures and public cryptographic keys. Indeed, permissions are usually distributed by encrypting secret keys with users' public keys (see Section 2). Therefore, compromising the integrity or availability of users' public keys or digital signatures would disrupt the CAC scheme by preventing the distribution or verification of new permissions. While preventing the CAC scheme to evolve, sensitive data are still intact, protected and available to all users (negligible impact for confidentiality and availability).

However, we identify a special case in which a user's (or role's) public key is not randomly tampered with, but it is instead replaced with another public key. In this case, all secret keys intended to be encrypted with the user's (or role's) public key are actually encrypted with another public key. Therefore, the owner of the corresponding private key (e.g., the attacker) would obtain all future permissions (e.g., assignments to roles or files) that the user (or role) would receive, with the potential to obtain access to (i.e., read and write) many files before being discovered (high impact for integrity).

6.2 Resources and Targets

We identify in Table 4c possible targets, i.e., where in the architectures a resource can be compromised. In general, a resource can be compromised at rest, in transit and in use. As in Section 5.4, we consider that the violation of the availability of a (used) resource does not affect the availability of the original (hosted) resource.

To define the targets, we recall the *Hosts* and *Uses* relations of Table 3d and 3e in Section 5.1; the *Hosts* relation defines which entity stores which resource (at rest), while the *Uses* relation (in use) implies a communication channel (in transit) between the entity hosting the resource and the entity using it. Therefore, a target can be either the entity storing the resource, the entity using the resource or the communication channel (denoted with ↔) between them. When only the confidentiality or the integrity properties can be compromised (i.e., the target is either a communication channel or an entity that uses the resource), we use the subscript *ci*. For instance, *Hosts* defines that encrypted data are hosted by the DS (first target), while *Uses* defines that encrypted data are used by both the proxy and the RM (two other targets). Since encrypted data need to be exchanged by these entities, we consider also the communication channels between the DS and the proxy or RM (two other targets). In total, encrypted data have five different targets. The same reasoning applies to metadata, hosted by the MS and used by the proxy and the RM. Instead, secret keys are hosted by the proxy and not used by any other entity. Therefore, the only target for secret keys is the proxy.

Finally, note that the presence of communication channels is conditional on the architecture; if two entities are in the same domain, the communication channel is implicit within that domain, i.e., data are not exchanged externally through the internet. Therefore, in this case, the communication channel (i.e., the target) does not exist.

6.3 Targets and Impact

We derive the relationship between targets and impact on sensitive data in Table 4d. We use Table 4c to identify which resources can be affected for each target. For instance, the proxy is a target for secret keys, metadata (only confidentiality and integrity) and encrypted data (only integrity). Then, we use Table 4b to determine the impact based on the affected resources. To be conservative, we derive the impact due to the violation of each security property by considering the worst-case. For instance, compromising the confidentiality of secret keys and metadata in the proxy has a high and negligible impact, respectively. Therefore, the worst-case impact is high.

6.4 Attackers and Architectures Exposure

We characterize attackers by complementing the trust assumptions formulated in Section 2.2 and reported at the beginning of this section:

- ($\neg$TA3$\lor\neg$TA4) *Malicious Insider*: we consider a malicious insider attacker (e.g., disgruntled employee) when the trust assumptions on domains (i.e., on-premise domain is fully trusted, CSP domain is partially trusted) do not hold. In this case, the on-premise and the CSP domains may be subject to internal attackers that may access and compromise any resource used or hosted by entities present in these domains;

- (¬TA1) *Man-in-the-Middle* (MitM): given that entities may be located in different domains, one or more communication channels must be established to allow for the exchange of data (e.g., *read* and *write* operations). However, if communication channels are not assumed to be secure (e.g., in an IoT context with limited cryptographic resources), an attacker may spoof or tamper with transferred data. We consider a MitM attacker for each pair of domains;
- (¬TA2) *Man-in-the-Device* (MitD): in enterprise contexts, especially in the last years, the Bring-Your-Own-Device (BYOD) paradigm has emerged in which employees use their own device also for work. However, this practice increments security risks related to the client$_u$ domain (e.g., viruses, side channels attacks). Also, unsupervised devices (e.g., IoT sensors) may be targeted by a physical attacker. Therefore, the MitD attacker affects the client$_u$ domain.

We can now compute the impact for each tuple architecture-attacker-security property. To illustrate our approach, we report some architectures with the related impact in Table 4e; the attacker determines which domain or pair of domains is affected, while the architecture indicates the possible targets. Depending on the architecture, the attacker may affect none, one or more targets. The impact level of an attacker is the worst-case impact of all targets affected by that attacker. For instance, in the first row of Table 4e, the malicious insider attacker in the CSP domain affects the MS, RM and DS entities. The impacts due to the violation of the CIA properties are, respectively, $\langle N, H, N \rangle$, $\langle N, H, N \rangle$ and $\langle N, L, L \rangle$. Therefore, the impact of this attacker on that particular architecture is $\langle N, H, L \rangle$. Note that the impact of the CSP malicious insider may change based on the architecture, as for the other attackers.

As mentioned at the beginning of this section, the likelihood of an attacker depends on the specific scenario and trust assumptions. Therefore, while we can precompute the impact, the likelihood is an input from the administrators. For instance, if an organization blindly trusts its employees, we should not consider the presence of a malicious insider in the on-premise domain. Also, if the organization has a strict policy on the security of the employees' devices (e.g., no BYOD, monitoring software, antivirus installed), the likelihood of a MitD attacker decreases. Once the likelihood is given, the risk level can be automatically computed as the product of impact and likelihood as reported in Table 4a.

Summarizing, in this section we have focused on the impact on sensitive data with respect to the violation of the CIA properties of the resources of CAC schemes. However, there are other aspects (e.g., redundancy, reliability) that are important to consider but do not depend on the trust assumptions reported at the beginning of this section; these will be discussed in Section 7.

7 TRADE-OFF ANALYSIS FOR ARCHITECTURAL DESIGNS

In this section, we formalize the problem (introduced in Section 3.3) of selecting the set of architectures $\mathcal{ARC}_{opt}$ that optimize both the risk levels and the multiple goals of a scenario as a Multi-Objective Combinatorial Optimisation Problem (MOCOP) [24].

Below, we first identify security and usability goals that may be desirable in different scenarios (as illustrated 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. We discuss how different architectures affect the degree of achievement of these security and usability goals. Then, we show how to reduce the problem of selecting the set of architectures $\mathcal{ARC}_{opt}$ that optimize both the risk levels and the goals into an optimization problem that considers the simultaneous maximization of two collections of objective functions; the first collection regards the minimization of the risk levels (or, equivalently, the maximization of the complement of the risk) of C, I and A, while the second collection regards the maximization of the goals. Finally, we propose two different algorithms to solve the MOCOP and study their computational complexity in the best, worst and average cases.

7.1 Identifying Goals

From both cloud-relevant literature and also from industrial-relevant technical reports [43], we sample 8 security and usability goals that may be desirable in our scenarios:

- *Redundancy* [21, 25, 44]: the extent to which the architecture allows to effectively have duplicated resources;
- *Scalability* [23, 25, 33, 41, 45, 46]: the ability of the architecture to scale up and down to accommodate dynamic workloads (e.g., the variable number of users' requests);
- *Reliability* [21, 23, 25, 35, 41, 45]: the ability of the architecture to keep working after the failure of one or more entities. We measure reliability through Single-Point-of-Failures (SPOF);
- *Maintenance* [21, 25, 33, 41, 44]: the easiness in the deployment and maintenance (i.e., software updates) of the architecture;
- *Denial-of-Service Resilience* [12, 23, 25, 35]: the intrinsic resilience of the architecture to Denial-of-Service (DoS) attacks;
- *Minimization of CSP Vendor Lock-in* [23, 45]: the easiness in switching CSP in the architecture (e.g., from AWS to Azure);
- *On-premise Monetary Savings* [21, 23, 25, 33, 45]: the monetary savings due to *not* adding entities to the on-premise domain;
- *CSP Monetary Savings* [21, 23, 25, 33, 45]: the monetary savings due to *not* adding entities to the CSP domain, e.g., because the organization already has an internal infrastructure.

7.2 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 7.1. Each entity-domain pair may have either a positive (+), negligible (=) or negative (-) effect on a goal.

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	=	=	=	+	-	+	=	-	+	=	-

- *Effect on Redundancy.* The CSP is a geographically distributed domain with mechanisms for replicating data across large areas [7]. On the contrary, the on-premise domain 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 [7]. The more entities are in the 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 [7]. 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.* The CSP domain is intrinsically resistant to DoS attacks [7]. 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, as discussed in Section 4, CAC may be unnecessary in architectures not using the CSP.

In contrast, hybrid architectures tend to balance the pros and cons. 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).[13] 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 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 affect 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, the organization can easily tune the effects in Table 3 based on its specific needs.

7.3 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 5.4, 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 *Pre-Filters* set containing the architectures that shall be excluded from the MOCOP. Consequently, we define $\mathcal{ARC}_{sub} \subseteq \mathcal{ARC}$ as the set of architectures in $\mathcal{ARC}$ but not in *Pre-Filters*:

$$\mathcal{ARC}_{sub} = \{arc \in \mathcal{ARC} | arc \notin \textit{Pre-Filters}\}; \tag{4}$$

[13] https://aws.amazon.com/s3/pricing/

7.4 Multi-Objective Combinatorial Optimization Problem

We consider the simple approach of combining 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 considering an objective function $g : \mathcal{ARC}_{sub} \mapsto \mathbb{Z}$ associated with each goal. Having as input the set of $\langle ent, dom \rangle$ pairs of an architecture $arc \in \mathcal{ARC}_{sub}$, 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.

To be conservative, we aggregate the 18 risk levels associated with an architecture (from Section 6) by assuming the worst risk level across all the attackers for each CIA property. For this, we add three more objective functions, namely g_C, g_I, g_A. Since we are considering the maximization of the objective functions, we make g_C, g_I and g_A measure the complement of the risk (i.e., the protection level) of each CIA property. To calculate the protection levels, we simply invert the scale of risk levels and associate a number from 0 (negligible protection, high risk level) to 3 (high protection, negligible risk level).

We observe that the Multi-Objective Optimisation Problem (MOOP) we introduced in [3] belongs to a particular sub-class of MOOP called MOCOP [24], whereby the set of possible solutions is discrete and is endowed with a particular algebraic structure. Indeed, the set $\mathcal{ARC}_{sub}$ of candidate architectures is finite and it is structured according to the relations introduced in Section 5 (Figure 3a). When considering cloud-based CAC schemes only, $\mathcal{ARC}_{sub}$ is also bounded (as already observed in Section 5.4, it contains 81 elements). However, we highlight that the architectural model presented in Section 5 can be extended to other paradigms as well (e.g., IoT, Edge Computing). In turn, this would allow us to identify further domains, entities and resources (e.g., publish/subscribe IoT brokers, intermediate edge computing nodes), thus expanding the set $\mathcal{ARC}_{sub}$. As we explain below, these observations allow us to reuse available techniques for the solution of instances of the MOCOP to perform the trade-off analysis necessary to identify the best candidate architectures for a given deployment. We highlight that, when considering only the three objective functions g_C, g_I, and g_A in the MOCOP, the trade-off analysis reduces to a risk assessment, i.e., to the problem of finding the most secure architectures for a given scenario. Instead, when considering only goals, the trade-off analysis reduces to the problem of finding the most performant architectures for a given scenario.

We are now ready to formalize as a MOCOP [24] the problem of finding the set of architectures $\mathcal{ARC}_{opt} \subseteq \mathcal{ARC}_{sub}$ such that the tuple $(g_{\text{Redundancy}}, \ldots, g_C, g_I, g_A)$ of objective functions measuring the degree of achievement of the goals and the protection levels (see Table 4 for an example) is maximum:

$$\max_{arc \in \mathcal{ARC}_{sub}} \; (\; g_{\text{Redundancy}}(arc), \ldots, g_C(arc), g_I(arc), g_A(arc) \;); \tag{5}$$

Particular care should be put in the definition of the operator max as it may be impossible to find a solution (a single architecture, in our case) that simultaneously maximizes all objective functions in (5). In fact, for any non-trivial MOCOP, there is no single solution that is simultaneously optimal for every goal (e.g., see [24]). Instead, there may exist several architectures (i.e., $\mathcal{ARC}_{opt}$ is a finite set of cardinality equal or larger than 1) that can be considered equally good, called Pareto Optimal [24]. Formally, an architecture $arc^* \in \mathcal{ARC}_{sub}$ is *Pareto Optimal* if and only if there is no architecture $arc \in \mathcal{ARC}_{sub}$ such that $g_i(arc) \geq g_i(arc^*)$ for each g_i in the set of objective functions G and $g_j(arc) > g_j(arc^*)$ for at least one $g_j \in G$. In other words, an architecture is Pareto Optimal if there does not exist another architecture that improves one objective function without detriment to another. In this case, following the notation in [14], we write $arc^* \succ arc$ to denote that arc^* ties or is higher than *arc* on each objective function and it is strictly higher on at least one, i.e., arc^* *dominates arc*.

Since the set $\mathcal{ARC}_{sub}$ of possible solutions is finite (recall that $\mathcal{ARC} \supseteq \mathcal{ARC}_{sub} \supseteq \mathcal{ARC}_{opt}$), it is possible to enumerate all vectors of objective function values and then find those satisfying the definition of Pareto optimality; the decidability of (5) is thus obvious. Below, we discuss two possible approaches to solve the MOCOP, i.e., the definition of a set of Pareto Optimal architectures (Section 7.5) and the reduction of (5) to a single objective optimisation problem (Section 7.6).

7.5 Solving the MOCOP - Multi-Dimensional Maximum Vector Problem

One straightforward approach to solve (5) consists in transforming the MOCOP into the multi-dimensional maximum vector problem [14] that amounts to identifying the maximals over a collection of vectors. The idea is to consider the vectors of values obtained by evaluating the objective functions on all the architectures in $\mathcal{ARC}_{sub}$ (e.g., see the rows in Table 4 as vectors) and then reuse off-the-shelf the available algorithms for solving the multi-dimensional maximum vector problem (e.g., those in [14]). In this way, we do not exploit the algebraic structure of the set $\mathcal{ARC}_{sub}$ as suggested in [24] to develop appropriate heuristics for improved performance. However, this is acceptable in our context in which the cardinality of $\mathcal{ARC}_{sub}$ is bounded by 81 (Section 5.4), although dedicated heuristics should be defined when extending the architectural model to arbitrarily complex scenarios (e.g., IoT, Edge Computing) in which the cardinality of $\mathcal{ARC}_{sub}$ (and $\mathcal{ARC}_{opt}$) can be assumed to be unbounded. We highlight that the number of objective functions can be assumed to be unbounded as well, since further goals can be considered besides the ones presented in Section 7.1. Below, we discuss the complexity of solving (5) with the multi-dimensional maximum vector approach.

Let $k = |G|$ be the number of objective functions, $n = |\mathcal{ARC}_{sub}|$ the number of possible solutions (i.e., architectures) and $p = |\mathcal{ARC}_{opt}|$ the number of Pareto Optimal solutions. The first step to solve (5) is to evaluate all objective functions for each architecture; this is easily seen to have complexity $O(k \cdot n)$. Then, to solve the resulting multi-dimensional maximum-vector problem, we use the `Best` algorithm described in [14] (and shown as Algorithm 1): `Best` first iterates over the set of possible solutions $\mathcal{ARC}_{sub}$ to find a *single* Pareto Optimal solution arc^*. Then, `Best` removes from $\mathcal{ARC}_{sub}$ all solutions that are not Pareto Optimal with respect to arc^*. In following iterations, more solutions can be found. Finally, `Best` ends when $\mathcal{ARC}_{sub}$ is empty.

As `Best` iterates over $\mathcal{ARC}_{sub}$ once for each Pareto Optimal solution, the complexity depends on p. In the best case there is a single Pareto Optimal solution (i.e., $p = 1$) and `Best` iterates only twice on $\mathcal{ARC}_{sub}$ by making k comparisons for each $arc \in \mathcal{ARC}_{sub}$, with a complexity of $O(k \cdot n)$. The worst case, where every architectures is optimal (i.e., $p = n$), requires n iterations with a complexity

Table 4. MOCOP Formalization

	Domain										Complement of Impact Level		
Client	On-premise	Cloud	Redundancy	Scalability	Reliability	Maintenance	DoS Resilience	Vendor Lock-in	On-premise Savings	CSP Savings	C	I	A
	🖥 📁	🛡 🛢	+1	0	0	0	0	−1	0	−1	0	0	+2
🖥	🛢 📁	🛡 🛢	0	+1	+1	+1	+1	0	+1	0	0	0	+2
🖥	🛢 📁		−2	−1	−1	−1	−1	+2	−1	+2	0	0	+2

Algorithm 1: `Best` algorithm from [14]

```
Input: $\mathcal{ARC}_{sub}$
Output: $\mathcal{ARC}_{opt}$

let $\mathcal{ARC}_{opt} = \emptyset$; // Initialize the empty set of Pareto Optimal Solutions
while $\mathcal{ARC}_{sub}$ is not empty do
    let $arc^* = \text{shift}(\mathcal{ARC}_{sub})$; // Get the first architecture
    foreach $arc$ in $\mathcal{ARC}_{sub}$ do
        if $(arc \succ arc^*)$ then
            $arc^* = arc$; // Replace the Pareto Optimal Solution
    end
    add($arc*$, $\mathcal{ARC}_{opt}$); // Add the new Pareto Optimal solution to $\mathcal{ARC}_{opt}$
    foreach $arc$ in $\mathcal{ARC}_{sub}$ do
        if $(arc^* \succ arc)$ then
            remove($arc$, $\mathcal{ARC}_{sub}$); // Remove the architecture from $\mathcal{ARC}_{sub}$
    end
end
```

of $O(k \cdot n^2)$. Finally, in the average case (i.e., $1 < p < n$), the complexity is $O(k \cdot n \cdot p)$. Generally, the complexity of `Best` is the sum of $O(k \cdot n)$, to calculate the values of objective functions, i.e., the vectors, and $O(k \cdot p \cdot n)$, to compute $\mathcal{ARC}_{opt}$. Therefore, the final complexity is $O(k \cdot n \cdot p)$.

After running `Best`, an organization is still left with the task of choosing one among the optimal solutions in $\mathcal{ARC}_{opt}$. This requires some ingenuity by experts in security and in the other fields to which the objectives belong (e.g., performance and quality of service). A possible strategy is that each group of experts independently evaluate the alternative architectures in $\mathcal{ARC}_{opt}$. Then, they meet to reach a consensus on the architecture that strikes the best possible trade-off by using priorities on the various goals or using criteria that are difficult to formalize as an objective function to be included in the MOCOP (5). Assuming that an organization already defined the priorities among the various objectives, it is possible to reduce the MOCOP (5) into a single objective optimization problem that amounts to considering one objective function obtained by a linear combination of the objective functions in (5) where the coefficients are weights representing priorities. Below, we elaborate on this idea by mapping (5) into a single objective optimization problem.

7.6 Solving the MOCOP - Single-Objective Optimisation Problem

After having solved the multi-dimensional maximum vector problem described in Section 7.5, administrators have to decide by themselves which architecture among the Pareto Optimal ones best fits their scenario. To provide a further tool to help administrators, we give the possibility to translate the MOCOP (5) into a Single-Objective Optimization Problem (SOOP). Again, we note that this is one possible formalization, where also others are possible (e.g., Minimum-Cost Flow [18], Generalized Assignment Optimization Problem [31]). In the SOOP, the objective function to optimize is the weighted sum of the objective functions $g_{\text{Redundancy}}(arc), \ldots, g_C(arc), g_I(arc), g_A(arc)$. In other words, we construct an objective function of the form $\sum_{g_i \in G} w_{g_i} \cdot g_i$. The constants $w_{g_i} \in \mathbb{R}$'s are *weights* and model the importance of achieving a certain protection level or goal. Technically, it is necessary to assume weights to be positive for guaranteeing that the solution of the transformed problem belongs to the set $\mathcal{ARC}_{opt}$ of Pareto Optimal solutions [24]. While the multi-dimensional maximum vector problem returns a set of architectures $\mathcal{ARC}_{opt}$, the SOOP exploits the a priori knowledge encoded in the weights defined by administrators to return a single architecture arc^*, i.e., the one deemed to be the best among the Pareto Optimal architectures.

However, we note there are two semantically distinct metrics with respect to which evaluate an architecture, i.e., the protection levels and the security and usability goals. As a validation example,

Algorithm 2: AdHoc Algorithm

```
Input: $\mathcal{ARC}_{sub}$, ( $w_{\text{Redundancy}}(arc), \ldots, w_C(arc), w_I(arc), w_A(arc)$ ) for each $arc \in \mathcal{ARC}_{sub}$
Output: $arc_{\text{goals}}$, $arc_{\text{protection}} \in \mathcal{ARC}_{sub}$

let $array_{\text{goals}}$ = []; // Array of pairs ⟨weighted summed goals, architecture⟩
let $array_{\text{protection}}$ = []; // Array of pairs ⟨weighted summed protection levels, related architecture⟩

foreach $arc$ in $\mathcal{ARC}_{sub}$ do
    $array_{\text{goals}}$.push( ($w_{\text{Redundancy}} \cdot g_{\text{Redundancy}}(arc) + \ldots + w_{\text{CSPSavings}} \cdot g_{\text{CSPSavings}}(arc)$, $arc$) )
    $array_{\text{protection}}$.push( ($w_C \cdot g_C(arc) + w_I \cdot g_I(arc) + w_A \cdot g_A(arc)$, $arc$) )
end
let $arc_{\text{goals}}$ = max($array_{\text{goals}}$)[1] // The best architecture for goals
let $arc_{\text{protection}}$ = max($array_{\text{protection}}$)[1] // The best architecture for protection levels
```

we choose to keep separate these two metrics and formulate two different SOOPs. We highlight that this is one possible option, while others are available (e.g., normalize the two metrics by the respective maximum value and then calculate the weighted sum). Given the MOCOP in (5) and a tuple ($w_{\text{Redundancy}}, \ldots, w_C, w_I, w_A$) of weights, we derive the following two SOOPs:

$$\max_{arc_{\text{goals}} \in \mathcal{ARC}_{sub}} (w_{\text{Redundancy}} \cdot g_{\text{Redundancy}}(arc) + \ldots + w_{\text{CSPSavings}} \cdot g_{\text{CSPSavings}}(arc)); \tag{6}$$

$$\max_{arc_{\text{protection}} \in \mathcal{ARC}_{sub}} (w_C \cdot g_C(arc) + w_I \cdot g_I(arc) + w_A \cdot g_A(arc)); \tag{7}$$

When solved, these two problems identify the two architectures that are best with respect to performance and risk exposure. The two architectures are guaranteed to be Pareto Optimal for the original MOCOP problem (5) only if they coincide, i.e., they are the same architecture [24]. If the solutions are different, then administrators should manually find the best one for their scenario.

Since relying on scenario-specific priorities, we name the algorithm to solve the two SOOPs as AdHoc and show it as Algorithm 2. As in Section 7.5, the first step is the evaluation of all objective functions for each architecture, i.e., $O(k \cdot n)$. Then, objective functions are weighted and summed together to obtain two values (i.e., goals and protection levels) for each architecture; the complexity is again $O(k \cdot n)$. Finally, to find the two best architectures, we simply select the two respective maximum values with complexity $2 * O(n)$. Therefore, the final complexity is $O(k \cdot n)$;

Finally, we can compare the complexity of Best (Algorithm 1) and AdHoc (Algorithm 2). By comparing the two complexities, i.e., $knp > kn$, we note that the complexity of Best is greater when $p > 1$. This means that the two algorithms perform the same in the best case only, while in all other cases the AdHoc algorithm is faster. In the next section, we describe an implementation of the algorithms in a proof-of-concept application to assist administrators in the deployment of the best CAC schemes architectures for the eHealth Section 3.1 and eGovernment Section 3.2 scenarios.

8 ASSISTED DEPLOYMENT OF CAC SCHEMES

In this section, we present how our architectural model and the MOCOP (5) can be used to assist administrators in the deployment of CAC schemes architectures. The workflow is summarized in Figure 5 as a two-step deployment process. The idea is to provide administrators with a dashboard in which they can set pre-filters, trust assumptions and requirements on the risk levels and security and usability goals of their scenario. In the first step, the dashboard allows performing a thorough "What-if" analysis to carefully assess the protection levels and the security and usability goals of architectures in $\mathcal{ARC}_{sub}$. For this, we provide an automated MOCOP Solver which embeds both

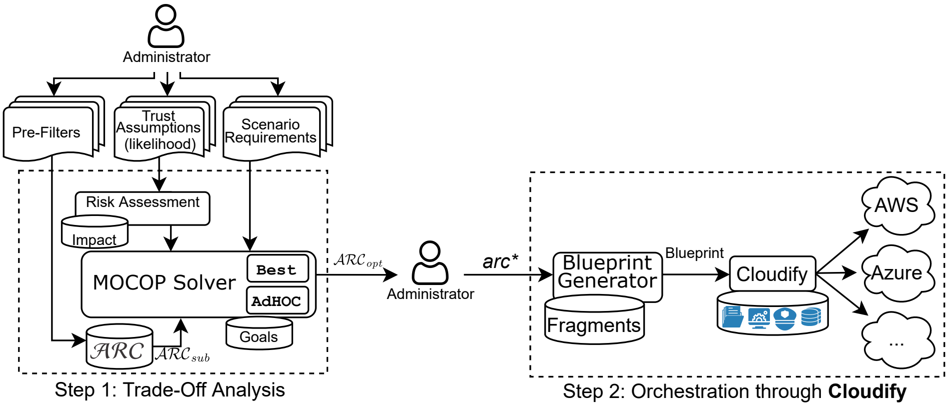


Fig. 5. Two-step Deployment Process

the Best and AdHoc algorithms. Through the What-if analysis, the administrators can find the most suitable architecture arc^* for their scenario. In the second step, the administrator inputs arc^* in the Blueprint Generator to automatically generate a deployable specification (Blueprint) based on the TOSCA[14] OASIS standard; a database containing blueprint fragments (Fragments) is used by the Blueprint Generator to build a TOSCA compliant representation of arc^*. We rely on the TOSCA-based Cloudify framework to automatically deploy the generated blueprint in the major CSPs, using the CAC entities we implemented (i.e., blue icons). The ultimate purpose is to optimize and simplify the time consuming and error-prone activity of manually selecting and deploying CAC schemes architectures in the cloud. As a final remark, in our deployment process we are not considering configuration tasks (e.g., the installation of the Cloudify framework software) or prerequisites (e.g., the subscription to the CSP to obtain administrator credentials) that are implicitly appointed to the organisation.

Below, we describe the dashboard and a proof-of-concept application of the two-step deployment process for the eHealth (Section 3.1) and eGovernment (Section 3.2) scenarios. We report the Blueprint fragments for the architecture arc^* of the eGovernment scenario in complementary material[15]. Finally, we describe the use of TOSCA and Cloudify to deploy the architecture arc^* for the eGovernment scenario (Section 8.4) and briefly discuss the implementation of a CAC scheme supporting this architecture (Section 8.5).

8.1 Web Dashboard

We implement both Best (Algorithm 1) and AdHoc (Algorithm 2) in a dashboard developed with web technologies (i.e., HTML, Javascript, CSS). Administrators can set pre-filters to define $\mathcal{ARC}_{sub}$ and assign a likelihood to each attacker. The dashboard allows administrators to decide which algorithm to use; if AdHoc is chosen, then administrators can provide the weights $w_{g_i} \in \mathbb{R}$ and decide whether to evaluate the architectures on the goals or on the protection levels. The dashboard solves the MOCOP (5) with the chosen algorithm and shows either the set $\mathcal{ARC}_{opt}$ of Pareto Optimal architectures (Best) or the single two best architectures (AdHoc) along with the details on the goals or protection levels.

In some cases, a constrained variant of the SOOP may be of interest. From the descriptions in Section 3.1 and Section 3.2, scenarios may benefit from the enforcement of hard and soft constraints. These are respectively mandatory and optional thresholds values expressing conditions

[14] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca
[15] https://stfbk.github.io/complementary/TOPS2020_2

Table 5. Performance Analysis of `Best` and `AdHoc` (time in milliseconds)

Case	`Best`	`AdHoc`	Metric
Best (p=1)	2.948	1.625	Goals (k=8)
		1.025	Protection Levels (k=3)
Average (p=40)	38.635	1.033	Goals (k=8)
		0.689	Protection Levels (k=3)
Worst (p=81)	131.702	0.682	Goals (k=8)
		0.672	Protection Levels (k=3)

on the single objective functions. When an (optional) threshold is undefined, we set its value to an arbitrarily large number; similarly, when a constraint associated with an objective is hard, we set the penalty value to an arbitrarily large number. For a given objective function g and associated threshold t and penalty v value, we define $[g]_v^t$ as follows:

$$[g]_v^t(arc) = \text{if } g(arc) < t \text{ then } g(arc) - v \text{ else } g(arc) \forall arc \in \mathcal{ARC}_{sub}; \tag{8}$$

Given a MOCOP (5), a tuple ($w_{\text{Redundancy}}(arc), \ldots, w_C(arc), w_I(arc) w_A(arc)$) of weights, a tuple ($t_{\text{Redundancy}}(arc), \ldots, t_C(arc), t_I(arc) t_A(arc)$) of threshold values and a tuple of penalty values ($v_{\text{Redundancy}}(arc), \ldots, v_C(arc), v_I(arc) v_A(arc)$), we derive the following two constrained variants of the original SOOPs:

$$\max_{arc \in \mathcal{ARC}_{sub}} (w_{\text{Redundancy}} \cdot [g_{\text{Redundancy}}]_{t_{\text{Redundancy}}}^{v_{\text{Redundancy}}}(arc) + \ldots + w_{\text{CSPSavings}} \cdot [g_{\text{CSPSavings}}]_{t_{\text{CSPSavings}}}^{v_{\text{CSPSavings}}}(arc)) \tag{9}$$

$$\max_{arc \in \mathcal{ARC}_{sub}} (w_C \cdot [g_C]_{t_C}^{v_C}(arc) + w_I \cdot [g_I]_{t_I}^{v_I}(arc) + w_A \cdot [g_A]_{t_A}^{v_A}(arc)); \tag{10}$$

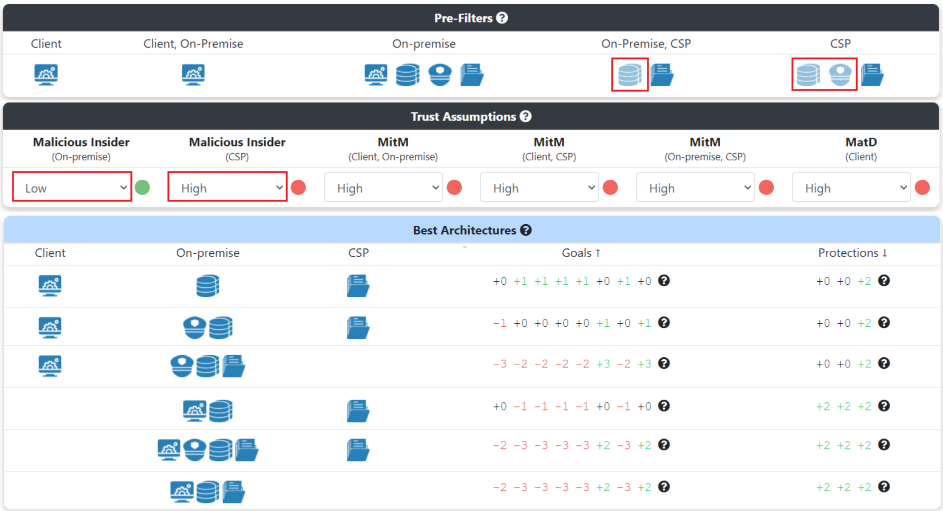
To give a clearer indication of the time needed to solve the MOCOP (5), we measure the efficiency (i.e., the run time) of the `Best` and (constrained) `AdHoc` algorithms in the best, worst and average cases. To do so, we manually tune the values of the objective functions to have only one Pareto Optimal architecture (best case), all Pareto Optimal architectures (worst case) and only half Pareto Optimal architectures (average case). For each case, we then run 1,000 iterations of the `Best` and the `AdHoc` algorithms to reduce possible measurement errors. We performed the tests on a Dell 7577 laptop equipped with an Intel(R) Core(TM) i7-7700HQ processor and 16GB of DDR4 RAM. We report the results in milliseconds in Table 5.

In the best case ($p = 1$), the time required to solve the MOCOP (5) by the `Best` and (constrained) `AdHoc` algorithms is still comparable, as the computational complexity for both is the same, i.e., $O(k \cdot n)$. As expected, the run time of `Best` increases as the cardinality of the set $\mathcal{ARC}_{opt}$ grows. In any case, we can argue that the actual time needed to solve the MOCOP is negligible (at most 131.702 milliseconds, less than one-eighth of a second). Below, we present two concrete applications to the eHealth (with `Best`) and eGovernment (with constrained `AdHoc`) scenarios.

8.2 MOCOP Application on the eHealth Scenario

As presented in Section 3.1, we suppose the presence of a clinic treating mental disorders. We report in Figure 6 a screenshot of the web dashboard configured to find $\mathcal{ARC}_{opt}$ for the eHealth scenario with the `Best` algorithm.

The "Pre-Filters" section allows administrators to set pre-filters to define $\mathcal{ARC}_{sub}$. For instance, the requirement eHR1 (need to hide metadata to avoid information leaking) makes metadata sensitive, excluding therefore architectures with the MS or RM entity in the CSP domain, i.e.,



Pre-Filters

Client	Client, On-Premise	On-premise	On-Premise, CSP	CSP

Trust Assumptions

Malicious Insider (On-premise)	Malicious Insider (CSP)	MitM (Client, On-premise)	MitM (Client, CSP)	MitM (On-premise, CSP)	MatD (Client)
Low	High	High	High	High	High

Best Architectures

Client	On-premise	CSP	Goals ↑	Protections ↓
			+0 +1 +1 +1 +1 +0 +1 +0	+0 +0 +2
			−1 +0 +0 +0 +0 +1 +0 +1	+0 +0 +2
			−3 −2 −2 −2 −2 +3 −2 +3	+0 +0 +2
			+0 −1 −1 −1 −1 +0 −1 +0	+2 +2 +2
			−2 −3 −3 −3 −3 +2 −3 +2	+2 +2 +2
			−2 −3 −3 −3 −3 +2 −3 +2	+2 +2 +2

Fig. 6. Web Dashboard for eHealth Scenario (Best Algorithm)

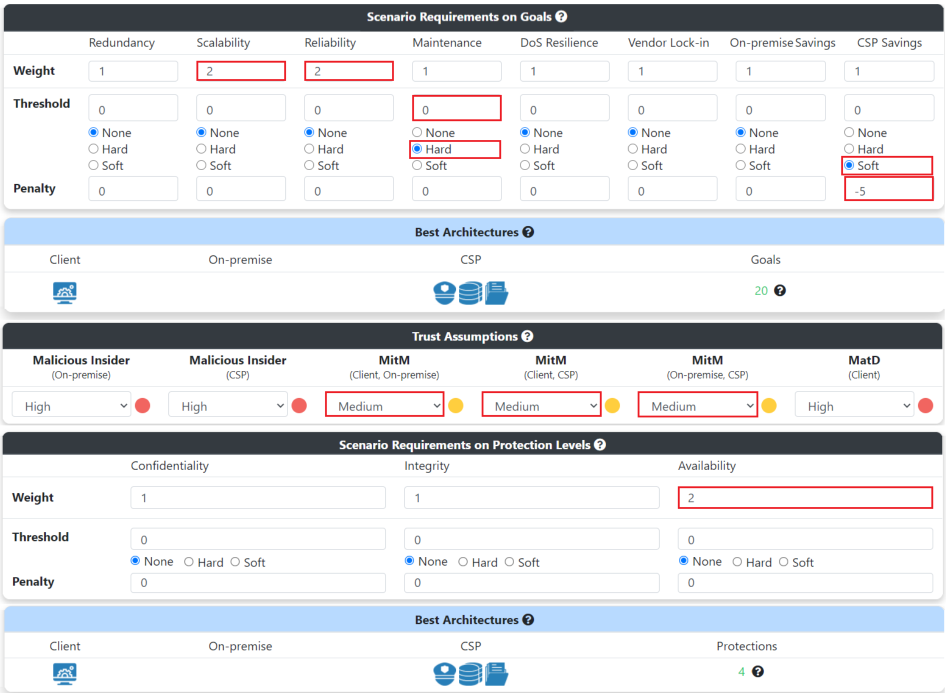
$\mathcal{ARC}_{sub} = \{arc \in \mathcal{ARC} | \langle MS, CSP \rangle \notin arc, \langle RM, CSP \rangle \notin arc\}$. As said in Section 5.1, metadata could be encrypted and turned into not-sensitive metadata at the cost of additional overhead on the CAC scheme. This would allow the MS to stay in the CSP domain. The RM, however, would still need to access plain-text metadata, so the RM could not stay in the CSP domain anyway. Considering a highly specialized clinic with well-paid doctors, eHA1 (there is a low probability of disgruntled doctors) lowers the likelihood of an on-premise malicious insider, for instance to low. Instead, for the same reasons explained in Section 3.1, eHA2 (CSP may be curious about medical data and metadata) suggests that a CSP may actively try to access medical record, suggesting a high likelihood for a CSP malicious insider.

The "Best Architectures" section in Figure 6 shows the resulting Pareto Optimal architectures. Having all zero or positive objective function values, the first architecture of the list may seem the optimal choice for the eHealth scenario. However, it also has zero protection level on the confidentiality and integrity of sensitive data. This is due to the high number of communication channels and the presence of the proxy entity in the client domain, which amplify the impact of MitM and MatD attackers. On the other hand, while having several negative objective function values, the three architectures at the end of the list are instead more secure, as they imply fewer communication channels and no entities in the client domain. Even though not identifying a single architecture for scenarios in which requirements are not known a-priori, the web dashboard allows organisations to investigate the trade-off between functionality and risk exposure to make more informed decisions through a "What-if" analysis.

8.3 MOCOP Application on the eGovernment Scenario

As presented in Section 3.2, 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 7 a screenshot of the web dashboard to find arc_{goals} and $arc_{\text{protection}}$ for the eGovernment scenario with the constrained AdHoc algorithm.

The requirement eGR1 (need to enable citizens' access from anywhere) may suggest storing data in a public cloud to allow for ubiquitous access to encrypted data. This implies excluding architectures with the DS entity in the on-premise domain, i.e., $\mathcal{ARC}_{sub} = \{arc \in \mathcal{ARC} | \langle DS, on\text{-}premise \rangle \notin arc\}$. With the AdHoc algorithm, administrators can set their requirements in terms of weights



Scenario Requirements on Goals

	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 ● None ○ Hard ○ Soft	0 ● None ○ Hard ○ Soft	0 ● None ○ Hard ○ Soft	0 ○ None ● Hard ○ Soft	0 ● None ○ Hard ○ Soft	0 ● None ○ Hard ○ Soft	0 ● None ○ Hard ○ Soft	0 ○ None ○ Hard ● Soft
Penalty	0	0	0	0	0	0	0	-5

Best Architectures

Client	On-premise	CSP	Goals
			20

Trust Assumptions

Malicious Insider (On-premise)	Malicious Insider (CSP)	MitM (Client, On-premise)	MitM (Client, CSP)	MitM (On-premise, CSP)	MatD (Client)
High ●	High ●	Medium ●	Medium ●	Medium ●	High ●

Scenario Requirements on Protection Levels

	Confidentiality	Integrity	Availability
Weight	1	1	2
Threshold	0 ● None ○ Hard ○ Soft	0 ● None ○ Hard ○ Soft	0 ● None ○ Hard ○ Soft
Penalty	0	0	0

Best Architectures

Client	On-premise	CSP	Protections
			4

Fig. 7. Web Dashboard for eGovernment Scenario (AdHoc Algorithm)

$w_i \in \mathbb{R}$ and constraints on the values of the objective functions of goals and protection levels. 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 higher weight to the scalability and reliability goals. For instance, the values can weight twice as much (i.e., $w_{g_{Scalability}} = 2$, $w_{g_{Reliability}} = 2$). In line with the requirement eGR1, we could set the weight of the availability property to two (i.e., $w_{g_C} = 2$). Finally, the trust assumption eGA1 (the presence of disgruntled employees is possible) suggests a high likelihood for on-premise malicious insiders. Moreover, the trust assumption eGA2 (communication channels may not always be secure) could set the likelihood of MitM attackers to medium.

We can see from Figure 7 that the two architectures arc_{goals} and $arc_{\text{protection}}$ for the eGovernment scenario coincide. The resulting architecture uses CSP services as much as possible. This reflects the scalability and reliability priorities. At the same time, this architecture enhances the simplicity of deployment and maintenance while avoiding the use of the on-premise domain (due to the high likelihood of a malicious insider).

8.4 Architecture Modeling with Cloudify

Once identified the most suitable CAC scheme architecture for a 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 reduce cost and time associated with the migration of cloud applications across different CSPs. For the actual modelling, we choose Cloudify,[16] a TOSCA-based cloud orchestration framework supporting major CSPs (e.g., Azure, AWS, Google, OpenStack). Cloudify allows graphical modelling of cloud applications by creating and configuring cloud services like servers and network appliances. The graphical model is translated to a TOSCA-compliant YAML 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 template of the most suitable CAC scheme architecture of the eGovernment scenario and report it in complementary material[17]. In detail, we model 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. Further blueprints corresponding to other architectures can be deployed as easily. Finally, we highlight that, while automating as much as possible the deployment of the CAC scheme architecture, some steps (e.g., the subscription to the CSP to obtain administrator credentials, the installation of Cloudify) cannot be automated and are appointed to the organisation.

8.5 CAC Scheme Implementation

As the last step to make our approach operational, we provide a fully working implementation of the CAC scheme proposed in [12]. We choose this scheme because it natively supports the architecture of the eGovernment scenario as well as many others. As explained in Section 2.2, the CAC scheme in [12] uses hybrid encryption; we choose RSA [38] for asymmetric encryption and AES for symmetric encryption.

We implement the proxy as a Java program, named CryptoAC, to be installed in the computer of each user. The users' secret key is generated and stored inside the user's proxy, which then allows the administrator to manage the AC policy and users to add, read and write files accordingly. We deploy the other entities (i.e., MS, RM and DS) using AWS services. In particular, we implement the MS as an AWS RDS MySQL database. To limit the disclosure of the AC policy to the users, we employ views and row-level permissions to automatically filter entries not associated with the user querying the database. In this way, each user knows his portion of the AC policy only and we respect the need-to-know principle (i.e., each user has access only to the information strictly needed to accomplish his task). Then, we implement the RM with Lambda functions and the DS with AWS S3. Through the Java AWS SDK, the proxy can invoke the RM to send add and write files requests and the DS to download (encrypted) files, respectively.

Finally, we test our implementation with several simulated sequences of operations involving 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 for the proxy based on web technologies[18] and made available RESTful APIs for a possible centralized use (i.e., proxy on-premise). We made CryptoAC open-source,[19] along with other resources.[20]

9 RELATED WORK

The topic of preserving the confidentiality of data hosted in the cloud has been widely addressed both in theory and practice. In this section, we report related work from both industry and research.

9.1 Cryptography in Remote Storage Through External Software

Many free and commercial services are available to protect data stored in the cloud. Below, we offer a comprehensive analysis of the most mature solutions currently available in the market.

[16] https://cloudify.co/ [17] https://stfbk.github.io/complementary/TOPS2020_2 [18] https://stfbk.github.io/complementary/TOPS2020_2
[19] https://github.com/stfbk/CryptoAC [20] https://stfbk.github.io/complementary/TOPS2020_2

Mega[21] offers cloud storage services accessible through the Mega open-source client. Each file is encrypted with an AES key derived from a user's password.

Similarly, AxCrypt[22] is an open-source software for encrypting personal data with AES keys derived from user-chosen passwords. Users can then share their data as long as the receiver has an AxCrypt account. AxCrypt also offers an online service to manage users' password.

Kruptos 2[23] encrypts personal files and synchronises them across multiple devices. It supports several remote storage services like OneDrive, Google Drive and Dropbox. Also, it allows users to share their files with anyone by packaging encrypted data along with a decrypting routine and sharing through an external channel the password used to derive the decrypting key.

Cubbit[24] takes instead a different approach, as it does not rely on traditional CSPs. In fact, Cubbit provides to each user the hardware (called "Cubbit Cell") for storing and managing their data. Besides claiming a smaller environmental impact, the main motivation behind this approach is the possibility of decentralizing data storage in a peer-to-peer network of Cubbit Cells to achieve high redundancy and fault-tolerance. In addition, Cubbit Cells encrypt users' data with a symmetric key that is in turn encrypted with the users' password. In this way, data are protected from external attackers and malicious insiders. While being functional for individuals, AxCrypt, Kruptos 2 and Cubbit cannot scale to accommodate the needs of an organization.

Qumulo[25] is a file data platform offering a software-defined file system running both on-premise and on CSP like AWS or Google Cloud. From version 3.1.5, Qumulo Core is featuring encryption at rest to preserve the privacy of sensitive data in on-premise clusters. Unfortunately, Qumulo still relies on cloud-side encryption for cloud clusters, allowing CSPs to access sensitive data.

Zadara[26] is an enterprise storage solution compatible with AWS, Google Cloud and Azure. Although Zadara offers encryption at rest, this is on a coarse-grained volume-by-volume basis. Moreover, this feature is disabled by default. A volume is encrypted with a randomly generated AES key, which is in turn encrypted with a master key derived from a user's password.

Boxcryptor[27] targets organizations by providing end-to-end encryption and data sharing at both user and group level. All data are stored in the data centre of Boxcryptor's company. Each user is given a pair of public-private keys, stored client-side and encrypted with a symmetric key derived from a password chosen by the user. The modification of the AC policy following the revocation of permissions is not specified in the technical report.[28]

LucidLink[29] simulates a Network Attached Storage (NAS) on users' devices, while data are stored in a CSP. Each user is provided with an RSA key pair; the public key is stored as metadata, while the private key is encrypted with a user's password. Each file is encrypted using AES-256 and the encryption key of the parent folder wraps each file key. The administrator gives to a user access to a folder or file by encrypting the key of the folder or file with the user's public key. Files are split into multiple segments to enhance performance, while metadata are synchronized by a central service. Again, the handling of revocation of permissions is not mentioned in the technical reports.[30]

Concluding, several tools are available to preserve the confidentiality of cloud-hosted sensitive data and enforce AC policies. However, such tools either target individuals (i.e., Mega, AxCrypt, Kruptos 2, Cubbit), provide coarse-grained AC (i.e., Zadara), do not protect data from the CSP (i.e., Qumulo) or have architectural constraints (i.e., Boxcryptor and LucidLink). While not having the same level of maturity, our solution addresses instead exactly these issues.

[21] https://mega.nz/ [22] https://www.axcrypt.net/ [23] https://www.kruptos2.co.uk/ [24] https://www.cubbit.io/technology
[25] https://qumulo.com/ [26] https://www.zadara.com/ [27] https://www.boxcryptor.com
[28] https://www.boxcryptor.com/en/technical-overview/ [29] https://www.lucidlink.com/ [30] https://www.lucidlink.com/security/

9.2 Cryptography in Remote Storage Through CSPs' Services

CSPs offer now several cryptography-based solutions to secure customers' data from external attackers. OneDrive features now a new security module, Personal Vault.[31] Besides enforcing strong authentication, Personal Vault offers encryption of data at rest and in transit. MongoDB, a popular database engine, started offering from version 4.2 a client-side field-level encryption feature[32] which preserves the confidentiality of data with respect to external attackers and also the server hosting the database by performing all encryption and decryption operations in the client.

These solutions offered by CSPs enhance the security of users' data through cryptography. However, they either perform cloud-side encryption, thus relying on a central trusted entity to mediate users' accesses (i.e., cloud-side AWS, OneDrive), or do not propose a mechanism to define and enforce AC policies (i.e., client-side AWS, MongoDB).

A different cloud-based solution to achieve secure and regulated data sharing could be the use of Hardware-based Trusted Execution Environments (TEE). A TEE is a secure area of the processor that guarantees the confidentiality and integrity of data and instructions through isolated execution. ARM TrustZone[33] and Intel SGX[34] are two instances of TEE that are now becoming available in major CSPs (e.g., Google Cloud[35], Azure[36]). In CAC, a TEE may be used as a tool to store cryptographic keys and encrypt and decrypt sensitive data. Kurnikov et al. [26] implemented a TEE-based Cloud Key Store (CKS) using password-based remote attestation to authenticate users and grant the use of cryptographic keys. Furthermore, CKS provides the possibility to define AC policies over keys so to share them among users. With respect to traditional CAC, CKS solves the issue of key revocation, as no user actually access any key. However, all cryptographic operations (i.e., encryption and decryption of files) occur within the TEE, possibly limiting the scalability of the approach. Indeed, it would be interesting to study the two approaches (i.e., traditional CAC like [12] and TEE-based CAC like [26]) to compare the relative performance in different scenarios.

9.3 Cryptographic Access Control

CAC has been applied in several scenarios, like local filesystems [17] and the cloud [12]. Goyal et al. [17] 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 [28], 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. [12] 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 likely to produce significant computational overheads. Besides, many other trust assumptions (e.g., presence of internal attackers) and usability goals (e.g., scalability, reliability) are often overlooked when designing a CAC scheme. Mainly, this is because a concrete deployment for a given scenario is seldom considered. These issues are instead thoroughly investigated in our solution.

9.4 Cryptographic Access Control Architectures

There are few works [13, 22, 34, 41, 45–47] that presented an architecture for their CAC scheme.

[31] https://www.microsoft.com/en-us/microsoft-365/blog/2019/06/25/onedrive-personal-vault-added-security-onedrive-additional-storage/

[32] https://docs.mongodb.com/manual/release-notes/4.2/

[33] https://developer.arm.com/ip-products/security-ip/trustzone

[34] https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html

[35] https://cloud.google.com/blog/products/identity-security/advancing-confidential-computing-with-asylo-and-the-confidential-computing-challenge

[36] https://azure.microsoft.com/en-us/solutions/confidential-compute/

In [41], 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 usability goals, 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 [47], 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 analysed other goals of the architecture. The authors implemented their CAC scheme just for analysing the performance of the `read` and `write` operations.

In [46], the authors proposed a CAC scheme enabling users' anonymous access to data stored in the CSP. The authors implemented a prototype interacting with AWS, providing an interface so that further CSPs can be supported. However, trust assumptions and usability goals were not considered. The architecture is fixed and cannot be modified to accommodate different scenarios.

In [45], the authors proposed and implemented a CAC scheme in a proof-of-concept tool 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 performance and monetary costs were analysed. However, each file is associated with a single policy, hindering the scalability and maintainability of the whole AC scheme. The authors also considered extending the scheme with ABAC, but no concrete design is given.

Ghita et al. [13] 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 [34], the authors propose a CAC scheme similar to the one presented in [12]. 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 all the encryption layers. The administrator can set a threshold to the number of encryption layers, after which a de-onioning procedure occurs. The authors implemented their scheme and obtained only slightly worse performances with respect to [12] (i.e., 7.2%) while being able to immediately block access to files by revoked users. Unfortunately, they did not discuss the monetary costs their onion mechanism incurs due to the many cryptographic operations performed by the CSP. Moreover, metadata are necessarily stored in the CSP, and the architecture was never discussed explicitly. Again, the implementation had the only purpose of measuring the efficiency of the CAC scheme.

In [22], 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.

Djoko et al. [8] proposed to cryptographically enforce AC policies in the cloud by leveraging TEEs at client-side. The authors developed NEXUS, a CSP-independent stackable filesystem mapping generic Application Programming Interfaces (APIs) exported by the TEE to the actual underlying storage platform. NeXUS can enforce discretionary access control policies over shared volumes (i.e., at directory level) by attaching cryptographically protected metadata (e.g., file system layout, cryptographic keys, AC policy) and encrypting each object in the volume with a symmetric key. Each symmetric key is in turn encrypted with the volume symmetric key called rootkey. Volume sharing happens by securely sending the volume rootkey to other users through SGX Remote Attestation. Even though users have access to the rootkey and could therefore decrypt the whole volume, the TEE, acting as a RM, can securely enforce the AC policy embedded in the metadata.

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 trust assumptions and goals or alternative architectures responsive to the requirements of a given scenario.

10 CONCLUSION AND FUTURE DIRECTIONS

In this paper, we proposed a methodology to find the most suitable architecture of CAC schemes for the trust assumptions and the requirements of different scenarios. First, we discussed centralized vs. decentralized AC to define under which conditions traditional AC suffices to protect sensitive data from both external attackers and partially trusted CSPs. Then, we identified common elements involved in the architecture of CAC schemes and provided an architectural model expressing the set $\mathcal{ARC}$ of the candidate architectures. We performed a risk assessment to identify the risk levels associated with different architectures, depending on the trust assumptions of the specific scenario. Then, we showed how to evaluate different architectures based on security and usability goals. To find the optimal architectures, we formalized a MOCOP so to leverage well-known techniques for Pareto optimality. We proposed two different algorithms to solve the MOCOP and defined their computational complexity. For concreteness, we gave a proof-of-concept application of how the architectural model and the MOCOP can be used to assist administrators in the deployment of CAC schemes architectures. We implemented a web dashboard to solve the MOCOP and perform a "What-if" analysis on the resulting architectures to carefully assess the trade-offs of the protection levels and the security and usability goals. Furthermore, we evaluated the efficiency of the two algorithms in the dashboard and showed how the results are returned nearly in real-time. We used the TOSCA OASIS standard and the Cloudify framework to automatize the deployment of the architecture for the eGovernment scenario. Finally, we implemented a CAC scheme supporting this architecture and provide a fully working prototype with AWS. As a final remark, we highlight that our contributions are completely independent of the underlying CAC scheme and AC policy model.

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 like eBusiness, eBanking and Fin-Tech. Besides, we could design an extended architectural model to consider other paradigms besides cloud, like IoT and Edge Computing. Another interesting improvement would be extending our tool to support additional blueprints associated with more CAC schemes. Also, once defined a specific CAC scheme, its assets and operations, it would be interesting to perform a more fine-grained and formal analysis to automate the risk assessment on the CIA properties of resources to explore the impact on sensitive data. Finally, it would be interesting to compare the performance of traditional and TEE-based CAC in different scenarios.

Acknowledgements. We would like to thank the reviewers for their comments and efforts towards improving this work. Stefano Berlato, Roberto Carbone and Silvio Ranise were supported in part by the Integrated Framework for Predictive and Collaborative Security of Financial Infrastructures (FINSEC) project that received funding from the European Union's Horizon 2020 Research and

Innovation Programme under Grant agreement no. 786727. Adam J. Lee was supported in part by the National Science Foundation under awards CNS–1253204 and CNS–1704139.

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] Stefano Berlato, Roberto Carbone, Silvio Ranise, and Adam J. Lee. Exploring architectures for cryptographic access control enforcement in the cloud for fun and optimization. In *15th ACM ASIA Conference on Computer and Communications Security (ASIACCS 2020)*. ACM, 2020.

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

[5] 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.

[6] 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.

[7] 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.

[8] Judicael B. Djoko, Jack Lange, and Adam J. Lee. NeXUS: Practical and secure access control on untrusted storage platforms using client-side SGX. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 401–413. IEEE.

[9] 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.

[10] Anna Lisa Ferrara, Georg Fachsbauer, 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.

[11] 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.

[12] 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.

[13] 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.

[14] Parke Godfrey, Ryan Shipley, and Jarek Gryz. Algorithms and analyses for maximal vector computation. *The VLDB Journal*, 16:5–28, 01 2007.

[15] S. Goel and V. Chen. Information security risk analysis – a matrix-based approach. 2005.

[16] 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.

[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*, pages 89–98, 01 2006.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] Kathrin Klamroth. Discrete multiobjective optimization. In Matthias Ehrgott, Carlos M. Fonseca, Xavier Gandibleux, Jin-Kao Hao, and Marc Sevaux, editors, *Evolutionary Multi-Criterion Optimization*, volume 5467, pages 4–4. Springer Berlin Heidelberg. Series Title: Lecture Notes in Computer Science.

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

[26] Arseny Kurnikov, Andrew Paverd, Mohammad Mannan, and N. Asokan. Keys in the clouds: Auditable multi-device access to cryptographic credentials. In *Proceedings of the 13th International Conference on Availability, Reliability and Security - ARES 2018*, pages 1–10. ACM Press.

[27] 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.

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

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

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

[41] Hiroyuk 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.

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

[43] Enrico Signoretti. GigaOm radar for file-based cloud storage. Last Modified: 2020-06-11 Library Catalog: gigaom.com Publisher: Gigaom.

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

[45] Yang Tang, Patrick P.C. Lee, John C.S. Lui, and Radia Perlman. Fade: Secure overlay cloud storage with file assured deletion. *IEEE Transactions on Dependable and Secure Computing*, 9(6):903–916, November 2012.

[46] 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.

[47] 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.