UNIVERSITY OF GENOA

PH.D. PROGRAM IN
SECURITY, RISK, AND VULNERABILITY
XXXVI CYCLE

Curriculum: *CyberSecurity and Reliable Artificial Intelligence (CSRAI)*

# A Security Service for Performance-Aware End-to-End Protection of Sensitive Data in Cloud Native Applications

by

**Stefano Berlato**

May, 2024

*Supervisors* Prof. Silvio Ranise, *University of Trento*
Dr. Roberto Carbone, *Fondazione Bruno Kessler*

*Ph.D. Coordinator* Prof. Alessandro Armando, *University of Genoa*

*Ph.D. Curriculum Coordinator* Prof. Luca Oneto, *University of Genoa*

*External Reviewers* Prof. Maribel Fernandez, *King's College London*
Prof. Alberto Trombetta, *Università degli Studi dell'Insubria*

Dibris

Department of Informatics, Bioengineering, Robotics and Systems Engineering

*“Da soli è un viaggio e basta, ma le avventure, quelle vanno condivise”*

# Abstract

The characteristics of cloud native applications — like decentralized architectures, high automation, and dynamic and interconnected microservices — bring forth a number of security challenges across both architectural design and lifecycle management. Some prominent challenges are authentication and authorization, real-time detection of security incidents, network security, microservice (as well as container) security, and, especially, *data security*. An ecosystem of security mechanisms already exists and provides excellent solutions addressing these challenges throughout the developing and operating of cloud native applications: identity and access management, monitoring and logging, intrusion prevention and detection systems, vulnerabilities assessment and hardening, and cryptography, to mention a few. Nonetheless, despite the availability of such a rich ecosystem, some cloud native applications entail additional considerations linked to the aforementioned challenges — and, in particular, to data security — which may need to be contemplated when evaluating the adoption of security mechanisms and their effectiveness. First, the level of trust assigned to participating parties within the scope of some cloud native applications is inherently limited — e.g., those aligning with the well-known security-by-design and zero trust principles. These cloud native applications confront a multifaceted threat landscape that extends beyond external attackers by including malicious insiders and honest-but-curious cloud providers which threaten the confidentiality and integrity of the (often sensitive) data managed by cloud native applications. Moreover, cloud native applications are frequently deployed in resource-constrained environments — e.g., the Internet of Things (IoT) — or operate in delicate fields (e.g., eHealth, automotive) offering critical functions (e.g., remote monitoring, cooperative vehicle maneuvering) where the quality of service may suffer from computationally or network heavy security mechanisms. In other words, security is not absolute, and its achievement must instead be balanced with that of performance requirements relevant to the underlying cloud native applications — e.g., low latency, minimal bandwidth utilization, and high scalability — underscoring the necessity for nuanced security mechanisms that are mindful of performance aspects. Therefore, in this thesis, we propose a security service addressing the convoluted dynamics of data security in cloud native applications. Our security service comprises four security mechanisms — namely `CryptoAC`, `ACE` and `ACME`, and `MOMO` — which implement the actual contributions of this thesis as we describe below. First, the threat model of cloud native applications requires preventing unauthorized access to data while offering strong guarantees of data confidentiality and integrity. To this end, we consider the use of cryptography to enforce Access Control (AC) policies — a combination usually called Cryptographic Access Control (CAC) — and propose the design of two CAC schemes, compatible with the aforementioned characteristics, for the end-to-end (E2E) protection of data both in transit and at rest in cloud native applications. We implement both CAC schemes — one for Role-Based Access Control (RBAC) and one Attribute-Based Access Control (ABAC) — into `CryptoAC` , discuss its security, and conduct a thorough performance evaluation. Then, we propose a methodology for evaluating the performance of generic AC enforcement mechanisms — hence, applicable to both CAC and centralized AC — starting from realistic workloads expressed as Business Process Model and Notation (BPMN) workflows. In detail, our methodology comprises a procedure deriving sequences of AC requests (e.g., access data, distribute permission) which are representative of the scenarios in which a cloud native application is deployed, and an evaluator executing these sequences against the AC enforcement mechanisms under test; we implement the procedure and the evaluator into `ACE` and `ACME`, respectively. Finally, we define an architectural model that identifies the common base building blocks of CAC over which we formalize a Multi-Objective Combinatorial Optimization Problem (MOCOP) to balance the achievement of security and performance in cloud native applications. Consequently, we implement an algorithm to solve the aforementioned MOCOP in `MOMO`, for which we provide both a conceptual application and a proof-of-concept application.

# Table of contents

# List of figures

# List of tables

# Chapter 1

# Introduction

The following paragraphs provide the reading key for the rest of Chapter 1 with the objective of showing how the contributions of the present thesis relate to each other by contextualizing them within the concepts of cloud native and DevOps; further details on the contributions themselves are provided in Chapters 2 to 4.

**Context — Digital Transformation and the Cloud.** Digital technologies have become ubiquitous; as a matter of fact, the capillary diffusion of intelligent devices — like laptops, smartphones, and Internet of Things (IoT) devices — and the proliferation of interconnected systems offer substantial opportunities to improve the well-being of society in several fields, including eHealth, Intelligent Transportation Systems (ITSs), industry 4.0, and smart buildings. In this setting, driven by the necessity of guaranteeing scalability, automation, and resiliency, software applications became closely related to cloud computing.[1] In particular, the cutting edge concepts of cloud native and DevOps — fostered also by major cloud providers like Amazon Web Services (AWS),[2] Google Cloud Platform (GCP),[3] and Azure[4] — are now crucial for the development and operating of software applications based on the cloud, offering a robust foundation for innovation, efficiency, and responsiveness in a rapidly evolving digital landscape. Indeed, cloud native and DevOps allow approaching such landscape from two different but complementary points of view — i.e., concerning architectures and lifecycles, respectively — which are both necessary to acquire a proper perspective and realize a comprehensive picture of cloud native applications — i.e., software applications developed and operated following the cloud native and (typically) the DevOps concepts.

**Problem — Multifaceted Challenges to Data Security in Cloud Native Applications.** While providing undeniable benefits, the widespread adoption of these digital technologies and concepts presents formidable challenges in upholding the security of the vast quantities of data being generated, exchanged, and stored by cloud native applications [4, 130]. These challenges are exacerbated by the high value of data — often of a sensitive or personal nature — as well as the large attack surface (e.g., due to the fragmentation into microservices [49]), the zero trust principle [101], the intricate threat model (see Section 1.3), and, most importantly, the delicate interplay between security and performance in cloud native applications [22]. As a consequence, challenges to data security in cloud native applications are naturally multifaceted. First, data are subject to a plethora of threats, such as

[1] The National Institute of Standards and Technology (NIST) defines cloud computing in [79] as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"

[2] What is Cloud Native? - Cloud Native Applications Explained - AWS (https://aws.amazon.com/what-is/cloud-native/)

[3] What Is Cloud Native | Google Cloud (https://cloud.google.com/learn/what-is-cloud-native)

[4] Cloud-Native Applications | Microsoft Azure (https://azure.microsoft.com/en-us/solutions/cloud-native-apps/)

external attackers, malicious insiders, and even well-intentioned yet overly inquisitive (i.e., *honest-but-curious*) cloud providers. The decentralized essence of cloud native applications — whereby often no fully trusted central party can be relied upon — makes it difficult to provide privileged access management to data effectively in the so-called Cloud-to-Thing Continuum [74], particularly since there may exist several data owners for different subsets of data [92]. Depending on the field (e.g., eHealth, automotive), the lack of data security may lead to disclosure and compromise of privileged information, disruption of critical functions, and potentially life-threatening situations. Secondly, we note that the adoption of security mechanisms for data protection in cloud native applications has a direct impact on the latter's quality of service, especially when security mechanisms are computationally demanding and the cloud native applications express stringent performance requirements. In other words, the dichotomy between security and performance — which notoriously conflict with each other [22] — calls for ad-hoc means to fine-tune the configuration of security mechanisms, with the final goal to strike the optimal trade-off balancing security and performance requirements proper to the considered cloud native application. Thirdly, fine-tuning such configuration requires the capability of precisely measuring the computational overhead incurred by a security mechanism in advance — that is, prior to its adoption. In particular, to produce truly representative (hence useful) results, we believe that the computational overhead of a security mechanism should be measured under realistic workloads specific to the cloud native application in which the security mechanism is adopted.

**Solution and Contributions — A Novel Security Service for Data Protection.** In this context, the present thesis proposes a security service offering a solution addressing the convoluted dynamics of data protection in cloud native applications. The security service comprises four security mechanisms: `MOMO` (Chapter 2), `ACE` and `ACME` (Chapter 3), and `CryptoAC` (Chapter 4). `CryptoAC` — short for *Cryptographic Access Control* — is an Access Control (AC) enforcement mechanism employing cryptography (which is ideal for decentralized environments) to both guarantee confidentiality and integrity of data as well as to specify and enforce fine-grained AC policies over data — a combination usually called Cryptographic Access Control (CAC). In other words, `CryptoAC` prevents unauthorized access while providing end-to-end (E2E) protection for data both in transit and at rest in cloud native applications. Intuitively, when compared to (traditional) centralized AC enforcement mechanisms, the use of cryptography in `CryptoAC` entails a computational overhead whose impact may not be readily quantifiable, especially in complex and dynamic scenarios — a scenario consists of an organization (e.g., hospital) offering a software application (e.g., remote patient monitoring) carrying out a set of business processes (e.g., collect medical data from IoT wearables and transmit them to the hospital for remote monitoring of patients) in a specific field (e.g., eHealth). For this reason, `ACE` and `ACME` — short for *AC state-change rule extraction procedurE* and *Access Control Mechanisms Evaluator*, respectively — constitute a methodology allowing an organization to measure and compare the performance (e.g., scalability, response time, and throughput) of different AC enforcement mechanisms — thus, also `CryptoAC` — and assess their suitability to its scenarios: `ACE` extracts sequences of activities (e.g., access to data, privileges distribution to users) that are realistic to the business processes carried out within the scenarios considered by an organization, while `ACME` allows for simulating the execution of such sequences of activities against an AC enforcement mechanism. The underlying intuition is that testing AC enforcement mechanisms under workloads which resemble those of actual production environments allows for producing truly significant performance results. Finally, each scenario may be characterized by distinct performance requirements (e.g., on latency and computational resources) and trust assumptions (e.g., on the presence or impact of certain security threats), resulting in a delicate interplay between performance and security that must be carefully evaluated when operating security mechanisms like `CryptoAC`.

For this reason, MOMO — short for *Multi-Objective Microservice Orchestration* — provides the means to identify on a case-by-case basis — that is, according to the underlying scenario — the optimal balance between the achievement of performance and security by configuring the deployment of CryptoAC accordingly, hence facilitating its integration with cloud native applications.

Below, we explain the complementary concepts of cloud native and DevOps (Sections 1.1 and 1.2, respectively) to clearly position the contributions of this thesis in the context thereof. Then, we describe the threat model for cloud native applications (Section 1.3) and identify the key needs for data security considered by this thesis (Section 1.4). Finally, we introduce the proposed security service, presenting its constituting security mechanisms, i.e., MOMO, ACE and ACME, and CryptoAC (Section 1.5), and outline the structure of the following chapters of this thesis (Section 1.6).

## 1.1 Cloud Native

The term *cloud native*[5] refers to the idea of developing and operating software applications following the principles of cloud computing — in particular, scalability, automation, and resiliency.[6] Cloud native is promoted by the Cloud Native Computing Foundation (CNCF),[7] founded by the Linux Foundation[8] in 2015 and dedicated to making cloud native ubiquitous by sustaining an ecosystem of open source and vendor-neutral software projects — such as Kubernetes (K8s)[9] and Prometheus[10] — for developing and operating cloud native applications. Cloud native applications are usually designed following the microservice architectural design, where a software application is decomposed into small, independent, and loosely coupled microservices. Each microservice performs a specific task and communicates with other microservices to achieve a certain common goal (e.g., carry out a business process) [4, 62]. The pattern for network communication among microservices may vary, but the two most commonly adopted patterns, often together, are synchronous request/response — usually through well-defined Application Programming Interfaces (APIs) over HTTP — and asynchronous publish/subscribe — usually through brokers managing event- and message-based channels [62]. The operating of microservices — involving the coordination and automation of various tasks related to, e.g., deployment, scaling, placement, and monitoring — is commonly called orchestration (see Section 1.1.1 for more details). To ensure consistency across deployments in different environments, microservices are typically issued as self-contained images, where an image is a read-only template containing instructions for creating a container — that is, a standalone executable package that contains software along with its dependencies.

### 1.1.1 The Cloud Native Stack

Devised by CNCF, the cloud native stack (see Figure 1.1) includes a combination of infrastructure, development, and operational technologies allowing to leverage the principles of cloud computing for software applications. The cloud native stack comprises 5 stacking layers:

[5] The CNCF style guide establishes that "cloud native" is not a compound adjective, hence it should be written without the hyphen and words should be lowercase (https://www.cncf.io/blog/2018/09/04/the-cloud-native-computing-foundation-cncf-style-guide/)

[6] It is worth noting that cloud native does not necessarily imply the use of public cloud providers, although it is usually the case

[7] Cloud Native Computing Foundation (https://www.cncf.io/)

[8] Linux Foundation (https://www.linuxfoundation.org/)

[9] Kubernetes (https://kubernetes.io/)

[10] Prometheus (https://prometheus.io/)

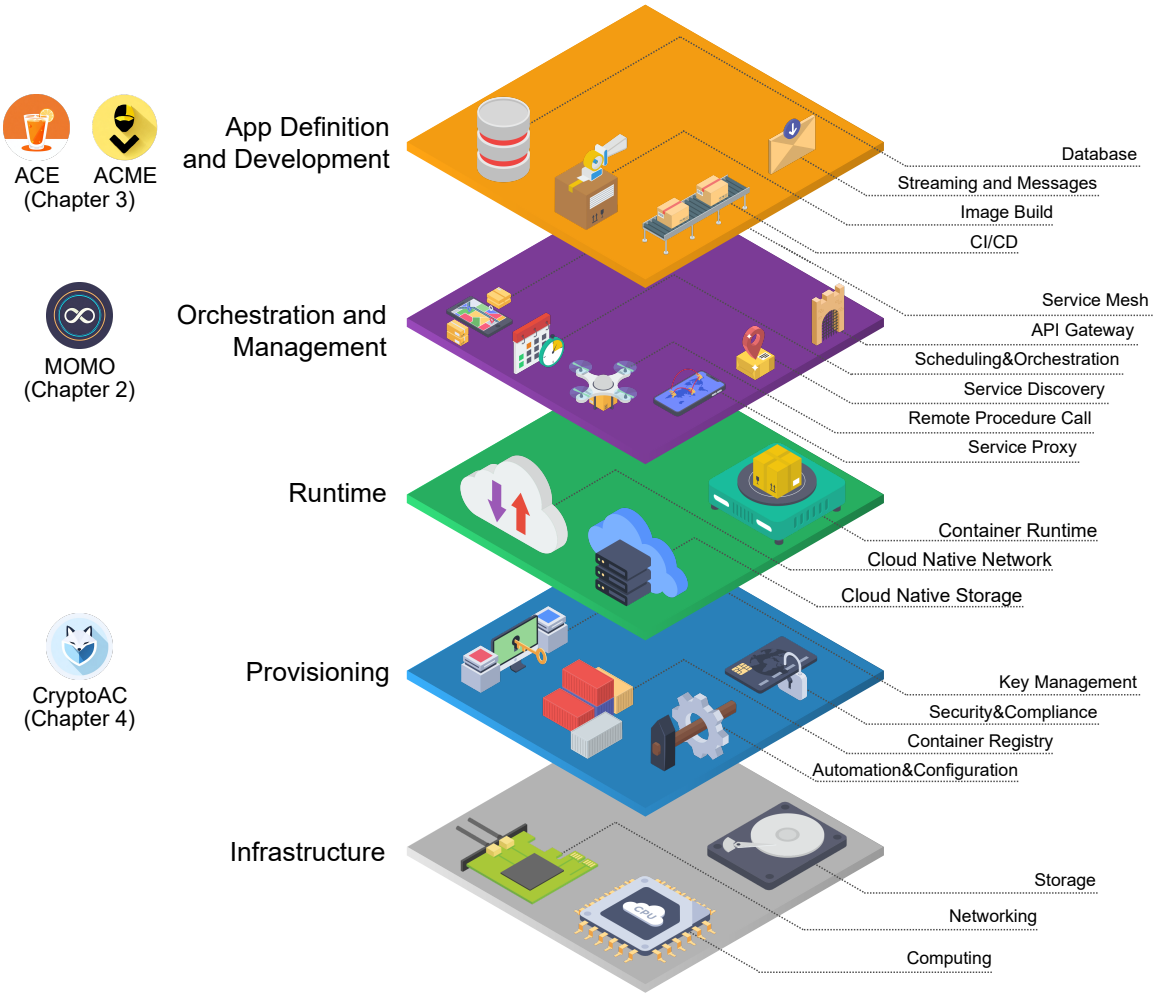


Figure 1.1: The cloud native stack of CNCF

- **Infrastructure**: this layer concerns the bare metal on top of which all software runs. The Infrastructure layer offers network and computational resources, storage, and operating systems, and it is usually leased from a (usually public) cloud provider;
- **Provisioning**: this layer groups technologies that help in provisioning the infrastructure — that is, in making the infrastructure usable by cloud native applications. The Provisioning layer allows for managing microservices — in terms of hosting and configuring images and containers — along with their security (e.g., hardening, key management) and compliance with policies and procedures;
- **Runtime**: this layer provides the environment in which cloud native applications run. The Runtime layer offers technologies to execute microservices, and manage the overlay networking — that is, virtual networks layered on top of existing network infrastructures, usually through Software-Defined Networking (SDN) techniques[11] — and the persistent storage enabling seamless movement of microservices across computing regions — a computing region is a specific geographical region offering a cluster of nodes (i.e., groups of physical or virtual machines) on which microservices can be deployed;
- **Orchestration and Management**: this layer abstracts the management of low-level aspects of microservices. The Orchestration and Management layer offers software to manage the

[11] What is software-defined networking (SDN)? | Cloudflare (https://www.cloudflare.com/learning/network-layer/what-is-sdn/)

scheduling and orchestration (e.g., placement), service discovery, APIs, proxying, and load balancing of microservices;

- **App Definition and Development**: this layer comprises software and tools supporting developers in the definition, implementation, and building of microservices and cloud native applications. The App Definition and Development layer offers software to assist during code writing and image building, along with toolsets providing the capability to set up Continuous Integration and Continuous Delivery/Deployment (CI/CD) pipelines (see Section 1.2.1 for more details on CI/CD).[12]

Besides the cloud native stack, CNCF considers two additional categories (which are orthogonal to the aforementioned 5 stacking layers): the **Platform** and the **Observability and Analytics** categories; the former groups vendors and platforms supporting the distribution, hosting, and installation of cloud native applications, while the latter considers software allowing to monitor, debug, and aggregate logs produced by cloud native applications.

## 1.2 DevOps

DevOps is a cultural and organizational approach in which (software) development and (information technology) operations teams work together to build, test, release, and deploy software in a shared responsibility fashion through seamless collaboration — indeed, the term DevOps is short for the words "development" and "operations", reflecting the idea of integrating these two aspects of software applications. DevOps allows for streamlining and accelerating the software development and operating lifecycle with high quality, responsiveness to changes, and efficiency. Notably, DevOps is also the approach through which cloud native applications are typically developed and operated — as said at the beginning of Chapter 1, cloud native and DevOps are strongly connected.

### 1.2.1 The DevOps Lifecycle

Figure 1.2 shows the DevOps lifecycle,[13] which is composed of 8 phases:

1. **Plan**: in this phase, relevant stakeholders (e.g., development and operation teams) meet to define goals (e.g., new features, bug fixes) and the steps toward the achievement of such goals;

2. **Code**: in this phase, developers write code, usually using version control systems (e.g., Git[14]), programming libraries, and collaborative tools such as cloud-hosted Integrated Development Environments (IDEs);

3. **Build**: in this phase, developers merge code changes into a central repository, triggering the execution of a suit of tests — such as integration and unit tests (i.e., white-box testing) — to identify issues and bugs as early as possible, and use automatic tools to compile and package code into a build;

[12] What is a CI/CD pipeline? (https://www.redhat.com/en/topics/devops/what-cicd-pipeline)

[13] At the time of writing, there is no general agreement on a standard DevOps lifecycle — as, intuitively, no lifecycle is suitable for all scenarios. Hence, in this thesis, we consider the lifecycle most widely adopted and recommended by, e.g., by Cisco (https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/devsecops-addressing-security-challenges.html), GitHub (https://resources.github.com/devops/pipeline/), and Atlassian (https://www.atlassian.com/devops). Note that, while other lifecycles might be more or less detailed or use different names for their phases, the underlying concepts usually remain the same

[14] Git (https://git-scm.com/)

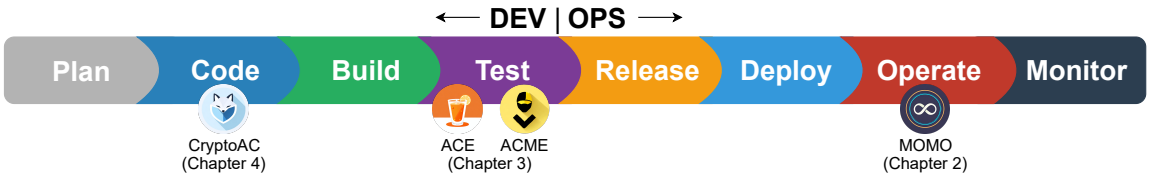


Figure 1.2: (One of) the DevOps lifecycle(s)

4. **Test**: in this phase, developers deploy the latest build in a testing environment which emulates the production environment as closely as possible. The latest build is then tested under different workloads to ensure the achievement of eventual performance requirements and identify any further bugs — usually through black-box testing. The Test phase is usually conducted automatically, although manual tests may occur to pinpoint specific issues. It is worth noting that the testing strategies and suites in the Test phase are usually tailored to the performance requirements of the organization and the software being tested (i.e., to the underlying scenario);

5. **Release**: in this phase, the latest build is finalized and scheduled for deployment into the cloud native application. The ultimate approval for deployment may be either manual or automatic — the latter is called *continuous deployment*;

6. **Deploy**: in this phase, the latest build is deployed into the cloud native application following either the creation of a new dedicated production environment — which immediately replaces the old production environment — or the so-called *blue-green* deployment, where the new and old production environments coexist for a predetermined amount of time — so to enable organizations to quickly rollback in case of issues in the new build or production environment;

7. **Operate**: in this phase, the cloud native application is orchestrated (e.g., microservices are placed into computing regions and nodes, scaled, and migrated) to meet the real-time demand of users and adapt to eventual modification of the underlying computing infrastructure (e.g., failure of a node);

8. **Monitor**: in this phase, the cloud native application is monitored to identify runtime issues (e.g., low uptime, the necessity for new features to support user behaviors). The monitoring of cloud native applications allows organizations to gather continuous feedback for the following Plan phases, closing the loop of the DevOps lifecycle.

**DevOps versus CI/CD versus Agile.** DevOps, CI/CD and Agile are interconnected but distinct concepts. As explained in Section 1.2, DevOps describes a cultural and organizational approach fostering collaboration between development and operation teams, emphasizing communication and responsiveness. On the other hand, CI/CD aims at automating the software development and operating lifecycle through well-defined and consolidated pipelines of tools. More in detail, continuous integration involves merging (and then testing) code changes into a central repository automatically. Then, continuous delivery automatically ensures that software is always ready to be deployed. Finally, continuous deployment involves deploying changes in production environments automatically. Agile, instead, is a software development process that emphasizes iterative development through small software releases and early integration of customers' feedback. As stated in the Agile manifesto,[15], Agile favors individuals and interactions over technologies and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to changes over

[15]Manifesto for Agile Software Development (https://agilemanifesto.org/)

following a plan. In summary, Agile sets the methodological foundation for the software development and operating process, DevOps ensures collaboration among teams throughout the software lifecycle (see Figure 1.2), and CI/CD provides the low-level automation, tools, and pipelines necessary to implement Agile and DevOps.

**DevOps and DevSecOps.** DevSecOps (short for the words "development", "security", and "operations") identifies the practice of introducing security throughout the DevOps lifecycle in an automated, organic, and traceable fashion; DevSecOps is also known as "shifting security left" or "security by design". Apart from this, DevSecOps considers the same lifecycle as DevOps (see Figure 1.2). At a high level (e.g., during the Plan phase), DevSecOps expects organizations to express security requirements on their cloud native applications concerning, e.g., vulnerability management (e.g., tracking and scanning) and network traffic analysis (e.g., inspect and filter network packets). Hence, organizations can define proper security policies — in terms of procedures and personnel management — satisfying the expressed security requirements and, consequently, identify the most suitable security controls — in terms of tools. For instance, security controls may include the use of security plugins for IDEs, Static Application Security Testing (SAST) (i.e., white-box testing in the Build phase), Dynamic Application Security Testing (DAST) (i.e., black-box testing in the Test phase), and real-time monitoring with, e.g., Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs). DevSecOps security requirements, policies and controls are often mapped to security frameworks such as MITRE ATT&CK,[16] benchmark standards such as those by the Center for Internet Security (CIS),[17] and automated tools implementing such benchmark standards like `kube-bench`,[18] respectively.

## 1.3 The Threat Model of Cloud Native Applications

The numerous advantages deriving from the adoption of cloud native and DevOps should be coupled with strong guarantees on the security of the involved data [4, 130], for cloud native applications operate under an intricate threat model (as we describe below). In addition, complex and dynamic scenarios, the decentralized nature of microservices, the large attack surface of cloud native applications, and the adoption of the zero trust principle all aggravate the problem of data security.

External attackers are the clearest and most intuitive example of a threat to data security, as demonstrated by the fact that they are considered by many widely adopted threat models such as Dolev-Yao [31] and Canetti-Krawczyk [20]. More in detail, Dolev-Yao formally expresses the capabilities of an external attacker having full control over the network [127]. In other words, the external attacker can eavesdrop, intercept, modify, and replay any network message. For instance, when transmitted over a network — a frequent event due to the fragmentation into microservices of cloud native applications — data are often vulnerable, among others, to eavesdropping, Man-in-the-Middle (MitM), and tampering attacks [49], especially if we consider that, in IoT, data are generally not protected with cryptography.[19] Data is vulnerable to external attackers also when at rest, as suggested by the steady increase in the number of cloud-related data breaches during the last years,[20] with dire consequences for the involved organizations; data breaches typically entail severe

[16] MITRE ATT&CK® (https://attack.mitre.org/)

[17] CIS Kubernetes Benchmarks (https://www.cisecurity.org/benchmark/kubernetes)

[18] GitHub - aquasecurity/kube-bench: Checks whether Kubernetes is deployed according to security best practices as defined in the CIS Kubernetes Benchmark (https://github.com/aquasecurity/kube-bench)

[19] 2020 Unit 42 IoT Threat Report (https://unit42.paloaltonetworks.com/iot-threat-report-2020/)

[20] ENISA Threat Landscape 2020 - Data Breach (https://www.enisa.europa.eu/publications/enisa-threat-landscape-2020-data-breach)

financial losses (e.g., direct expenses, fines) as well as reputational damage, with costs that can easily reach millions of dollars.[21]

In this regard, it should be acknowledged that all major cloud providers have a wide offering of security mechanisms to protect the data of organizations, such as centralized Identity and Access Management (IAM) and default cloud-side encryption (e.g., AWS,[22] GCP,[23] Azure[24]). Nonetheless, there are still some security concerns worth highlighting. For instance, cloud services for data storage may be misconfigured,[25,26,27,28] usually for lack of authentication or loose user permissions [137]. Cloud-side encryption may help in mitigating the consequences of data breaches, increasing the sense of security of organizations on the confidentiality of cloud-hosted data. Nevertheless, although organizations may trust cloud providers to properly handle and store data on their behalf, little can be done to preclude unauthorized access to such data by the cloud providers themselves. In fact, a report by the U.S. Federal Trade Commission states that cloud providers regularly collect data stored by organizations — e.g., to train Artificial Intelligence (AI) models, profile customers and users, or conduct market analyses — without the latter's knowledge [97]. The same reasoning applies to cloud services (e.g., AWS IoT Greengrass[29]) offering brokers that mediate (and can thus access the content of) communications happening among the microservices composing a cloud native application — e.g., through APIs and publish/subscribe channels (see Section 1.1). Notably, neither Dolev-Yao nor Canetti-Krawczyk considers possible threats coming from partially trusted parties (e.g., such as cloud providers).

Finally, internal attacks are not infrequent in cloud native applications [49]. In fact, the European Union Agency for Cybersecurity (ENISA) found that one of the most impactful attack vectors is the insider threat[30] — whether intentionally carried out by malicious insiders (e.g., disgruntled employees), coming from compromised microservices, or due to human errors. Again, neither Dolev-Yao nor Canetti-Krawczyk considers internal attackers.

In summary, the security of the (huge volumes of) data handled by cloud native applications faces numerous challenges, as their threat model presents a heterogeneous set of adversaries, encompassing external attackers, malicious insiders, and also honest-but-curious cloud providers (sometimes also referred to as *partially trusted* cloud providers). Nonetheless, as often considered in the literature [23, 87] the threat model of cloud native applications expects that attackers cannot tamper or compromise administration points, the initialization of services (e.g., the definition of public and private cryptographic parameters) or the registration and authentication of users.

[21] Cost of a data breach 2023 | IBM (https://www.ibm.com/reports/data-breach)

[22] Protecting data with encryption - Amazon Simple Storage Service (https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingEncryption.html)

[23] Default encryption at rest | Documentation | Google Cloud (https://cloud.google.com/docs/security/encryption/default-encryption)

[24] Personal Vault: Store Sensitive Files – Microsoft OneDrive (https://www.microsoft.com/en-us/microsoft-365/onedrive/personal-vault)

[25] The Scarily Common Screw-Up That Exposed 198 Million Voter Records (https://www.wired.com/story/voter-records-exposed-database/)

[26] Blame Human Error for WWE and Verizon's Massive Data Exposures (https://www.wired.com/story/amazon-s3-data-exposure/)

[27] Here's What It's Like to Accidentally Expose the Data of 230M People | WIRED (https://www.wired.com/story/exactis-data-leak-fallout/)

[28] Security News This Week: The Pentagon Left Data Exposed in the Cloud (https://www.wired.com/story/security-news-this-week-the-pentagon-left-data-exposed-in-the-cloud/)

[29] IoT Edge, Open Source Edge - AWS IoT Greengrass (https://aws.amazon.com/greengrass/)

[30] ENISA Threat Landscape 2020 - Data Breach (https://www.enisa.europa.eu/publications/enisa-threat-landscape-2020-data-breach)

## 1.4 Key Needs for Data Security in Cloud Native Applications

From Section 1.3, we can identify two main *needs* concerning data security in cloud native applications, namely (*i*) providing strong guarantees on the E2E confidentiality (i.e., the property for which a given resource can be accessed by authorized users only) and integrity (the property for which any alteration of a resource is immediately noticeable) of the involved data both when in transit and at rest — to thwart attempts to compromise or breach data from external attackers and malicious insiders — and (*ii*) mediating access to (especially sensitive or personal) data — to prevent unauthorized access by, e.g., malicious insiders and honest-but-curious cloud providers. The process of mediating access requests to data and determining whether requests should be granted or denied is called AC [106] and it is a fundamental component of any cloud native application [5]. More precisely, AC expects the presence of an AC policy (or simply "policy") declaring what users can perform what actions on what resources — where resources are logical vessels for data (e.g., files, documents); an AC policy is formally represented by an AC model (or simply "model") giving the semantics for granting or denying users' requests, while the software enforcing a policy based on the chosen model is called AC enforcement mechanism (or simply "mechanism"). In this regard, we note that traditional mechanisms relying on fully-trusted central parties might not always be suitable for data protection in (particularly zero trust-based) cloud native applications — as the amount of trust assigned to participating parties is limited by definition. Moreover, cloud native applications often present multiple data sources and owners [92], which are usually distributed throughout the network; hence, a centralized security mechanism may not be able to protect data in the Cloud-to-Thing Continuum effectively [74]. Therefore, developers may consider the adoption of mechanisms assuming minimal trust on the participating parties and operating in a decentralized fashion to protect data under the threat model discussed in Section 1.3. Accordingly, a security mechanism employing cryptography to provide CAC seems the natural solution to fulfill both of the aforementioned needs (i.e., E2E confidentiality and integrity and AC); indeed, CAC provides data confidentiality through encryption, data integrity through digital signatures and Message Authentication Codes (MACs), and enforcement of AC policies through a careful distribution of cryptographic keys. In other words, in CAC, data (e.g., files, messages) are encrypted, and the permission to access the encrypted data is embodied by the corresponding cryptographic decrypting keys — which are distributed to authorized users only by a single administrator. Hence, in CAC the administration of AC policies is centralized — avoiding synchronization and consistency issues — while their enforcement is pre-compiled into the (distribution of) cryptographic keys. The set of instructions describing how to distribute (or revoke access to) cryptographic keys is usually called CAC scheme; a CAC scheme is implemented by a CAC enforcement mechanism.

Although providing many benefits, CAC generally incurs non-negligible computational overhead due to the cryptographic computations involved. Thus, organizations have the *need* to carefully evaluate this computational overhead on their cloud native applications before adopting CAC enforcement mechanisms. Intuitively, generic performance evaluations are, by definition, not representative of a realistic use in a specific scenario; indeed, the performance of any AC enforcement mechanism is strictly related to its usage. A performance evaluation aiming at producing significant results — that is, results that are similar to those of a production environment — should be tailored to the scenarios considered by the organizations, and consider how a security mechanism would perform when tested under specific sequences of activities (e.g., data access, permission distribution) representative of such scenarios and their business processes.

Finally, the integration of CAC enforcement mechanisms with cloud native applications is not trivial since each scenario might present different performance requirements and trust assumptions that may even conflict with each other, highlighting the *need* of carefully analyzing compromises

and trade-offs; for instance, while elaborating data in the cloud promotes reliability and resilience to Denial of Service (DoS) attacks, it also increases network latency with respect to elaborating data at the edge of the network.

## 1.5 A Security Service for Cloud Native Applications

To address all the four *needs* identified in Section 1.4, in the present thesis we propose a security service providing a solution for the E2E protection and selective sharing of data that can be tested under realistic workloads and easily and optimally integrated into cloud native applications. Our security service offers high-level interfaces abstracting low-level details (e.g., configuration of cryptographic primitives) to be readily usable without requiring in-depth knowledge of the functioning of the security service itself. As mentioned at the beginning of Chapter 1, our security service comprises four security mechanisms which implement the actual contributions of this thesis, which we describe below:

- `MOMO` (described in Chapter 2). We define an architectural model that identifies the common base building blocks (also called *components*) of CAC, namely assets, entities, and domains: an asset is a category of information related to CAC (e.g., the set of all cryptographic decrypting keys), an entity is an active component that uses assets to execute some part of a CAC scheme (e.g., a software that uses cryptographic keys to en/decrypt data), while a domain is a logical or physical location that can host one or more entities. In other words, the architectural model allows for synthesizing all possible assignments of entities to domains — we refer to the set of entities-domains assignments composing a CAC scheme as *placement*. Afterward, we formalize an optimization problem to help organizations decide which placement for a CAC scheme — and the corresponding CAC enforcement mechanism — is the most suitable according to the performance requirements and trust assumptions of their scenarios. More in detail, we identify a set of objectives — where an objective is any property, goal, or quality of service desirable in a given scenario — and, for each objective, we propose an objective function to measure how much a certain placement attains (e.g., satisfies, fulfills) a certain objective. Then, we formalize as a Multi-Objective Combinatorial Optimization Problem (MOCOP) [65] the problem of identifying optimal placement(s), that is, placement(s) that maximize the values of all objective functions simultaneously — these optimal placements are usually referred to as the *Pareto Optimal* solutions to the MOCOP. Finally, we implement an algorithm to solve the aforementioned MOCOP by analyzing (apparently equivalent) optimal placements into a security mechanism called `MOMO`.[31] Briefly, `MOMO` takes as input the sets of entities and domains of a CAC scheme, as well as the set of objective functions relevant to the scenario being considered. Then, `MOMO` solves the MOCOP and identifies a single optimal placement. By mapping the high-level concepts of entity and domain — specified in our architectural model for CAC — to the concepts of microservices and computing regions — related to cloud native applications — respectively, `MOMO` translates the optimal placement previously identified into a concrete placement of microservices into computing regions. In other words, `MOMO` provides a technology that can be used within the Orchestration and Management layer of the cloud native stack (see Figure 1.1), as `MOMO` takes part in the orchestration of cloud native applications by guiding the placement of their constituting microservices. Correspondently, `MOMO` is used during the Operate phase of the DevOps lifecycle (see Figure 1.2) when the microservices composing a cloud native application are placed and deployed into computing regions and nodes. We also provide a proof-of-concept application

[31] MOMO UC-CCAM - FogAtlas-K8s (https://gitlab.fbk.eu/fogatlas-k8s/uc-ccam)

of MOMO by integrating it as a placement algorithm for K8s in the FogAtlas framework (please refer to Chapter 2 for more details);[32]

- ACE and ACME (described in Chapter 3). We propose a methodology allowing organizations to measure and compare the performance — in terms of scalability, response time, and throughput — of AC (and also CAC) enforcement mechanisms. To ensure a realistic performance evaluation and produce results significant to specific scenarios (and their business processes), we rely on Business Process Model and Notation (BPMN), which is the standard most commonly adopted by organizations to execute business processes represented as *workflows* [15, 104, 72] — in essence, a workflow is a sequence of activities. In case an organization does not use BPMN, we remark that there is an extensive body of literature on process mining for BPMN — that is, extracting BPMN workflows, usually from event logs (e.g., see [61]). Moreover, besides traditional business processes, we note that BPMN is also used to model (even automated) processes in emerging application fields (e.g., IoT-based scenarios) [77, 60]. Our methodology comprises two modules: (*i*) a procedure extracting sequences of activities (e.g., revoke a permission, access a piece of data) from workflows and (*ii*) a simulator executing these activities — reflecting the execution of the original workflows — on the AC enforcement mechanism under evaluation; we implement the procedure and the simulator into two security mechanisms called ACE[33] and ACME,[34] respectively. Following a well-established formal semantics (see [28]), ACE convert workflows into a subclass of Petri nets called WorkFlow (WF) nets — which are, essentially, graphs. Afterward, ACE computes the topological ordering of the WF net to identify all possible executions — that is, all sequences of activities allowing to achieve the business goal modeled in the original workflow. Finally, ACE translates the sequences of activities into sequences of AC requests — that is, invocations toward AC enforcement mechanisms (e.g., toward their APIs). These sequences of AC requests are then provided as input to ACME, which executes them to measure the performance of AC enforcement mechanisms. Notable, ACME supports the concurrent and intertwined execution of more sequences of AC requests corresponding to different workflows (i.e., business processes) to make the testing environment even more realistic. In other words, ACE and ACME provide technologies that can be used within the App Definition and Development layer of the cloud native stack, as ACE and ACME are part of the CI/CD pipeline (see Section 1.2.1) since they offer software to assist developers in testing new builds. Correspondently, ACE and ACME are used during the Test phase of the DevOps lifecycle when the latest build of a cloud native application is deployed in a testing environment and tested under different workloads which are tailored to the underlying scenario — and, therefore, also to its business processes. To demonstrate the effectiveness of our methodology, composed of ACE and ACME, we provide a proof-of-concept application on both two (centralized) AC enforcement mechanisms — that are Open Policy Agent (OPA)[35] and the AuthzForce Server (Community Edition) open-source implementation of eXtensible Access Control Markup Language (XACML)[36] — and one CAC enforcement mechanism — that is CryptoAC (see Chapter 4). We also compare the results obtained with ACE and ACME against results obtained with the approach typically used to conduct performance evaluations, i.e., micro-benchmarks (please refer to Chapter 3 for more details);

[32] FogAtlas (https://fogatlas.fbk.eu/)
[33] AC state-change rule extraction procedurE (https://github.com/stfbk/ACE)
[34] Access Control Mechanisms Evaluator (https://github.com/stfbk/ACME)
[35] Open Policy Agent (https://www.openpolicyagent.org/)
[36] AuthzForce Community Edition · GitHub (https://github.com/authzforce)

- `CryptoAC` (described in Chapter 4). As argued in Section 1.4, it is undeniable that CAC enhances data security in cloud native applications. Nonetheless, distinct scenarios may prefer different CAC schemes. More precisely, we note that there exist several well-known and consolidated AC models, each with its own advantages. For instance, Role-Based Access Control (RBAC) [107] and Attribute-Based Access Control (ABAC) [52] are two of the most widely adopted AC models [19]: RBAC provides simplicity and ease of implementation by assigning permissions based on predefined roles (e.g., job qualifications within an organization), while ABAC offers greater flexibility by allowing to evaluate AC requests based on, e.g., users' attributes as well as environmental conditions (e.g., date and time). Consequently, a scenario based on an enterprise context may prefer a RBAC policy — given the intuitive mapping between roles and job qualifications (e.g., employee, manager) — while a scenario set in a (more convoluted) IoT context may prefer a (more expressive) ABAC policy. For this reason, to provide a comprehensive security mechanism offering E2E protection of data in transit and at rest in cloud native applications for different scenarios, we design two different CAC schemes, one for RBAC and one ABAC, respectively: the former is derived from the work already presented in [39, 11], while the latter is a novel contribution. Besides security, both CAC schemes present a series of features to enhance their expressiveness and mitigate the computational overhead due to cryptographic computations, such as the capability of integrating CAC with traditional centralized AC enforcement mechanisms (see Section 4.2). We implement both CAC schemes into a security mechanism called `CryptoAC`,[37] an open-source, highly modular, and extensible tool written in the (multiplatform) Kotlin language[38] and designed as a microservice to guarantee seamless integration with cloud native applications. In other words, `CryptoAC` provides a technology that can be used within the Provisioning layer of the cloud native stack — where we find similar AC enforcement mechanisms like OPA — as `CryptoAC` relates to both key management and compliance with (AC) policies. Correspondently, `CryptoAC` is used during the Code phase of the DevOps lifecycle as, intuitively, developers need to write the code to invoke `CryptoAC`'s APIs in the microservices composing their cloud native applications. Finally, we discuss the security of `CryptoAC` with respect to a popular cryptographic-based security mechanism, i.e., Transport Layer Security (TLS), and conduct a thorough performance evaluation (please refer to Chapter 4 for more details).

`MOMO`, `ACE`, `ACME`, and `CryptoAC` are four security mechanisms designed to synergy with each other in the context of a single security service that, as a whole, yields an added value to both the cloud native (stack) and the DevOps (lifecycle) concepts, allowing organizations to provide better security for the data handled by their cloud native applications. Nonetheless, it is also true that these security mechanisms can be used independently of each other. For instance, given the definition of further objective functions suitable to different scenarios, `MOMO` can potentially be used as a placement algorithm for any cloud native application. Similarly, `ACE` and `ACME` allows organizations to measure and compare the performance of any AC enforcement mechanism, not only those based on CAC. Finally, although `MOMO`, `ACE`, and `ACME` enable organizations to optimally deploy CAC enforcement mechanisms while providing useful insights on their performance, `CryptoAC` can still be used as a standalone means to enhance data security in cloud native applications.

[37] CryptoAC is an open-source tool for the E2E protection of sensitive data through cryptographic enforcement of access control policies. (https://github.com/stfbk/CryptoAC)

[38] Kotlin (https://kotlinlang.org/)

## 1.6 Thesis Structure

As MOMO, ACE, ACME, and CryptoAC locate within different layers and phases of the cloud native stack and the DevOps lifecycle, each contribution of the present thesis requires different background notions — with few overlaps among each other. To streamline the reading experience and help the reader more easily browse through the following chapters, we do not collect and report all background notions within a single chapter. Conversely, we describe our contributions in self-contained chapters where background notions are introduced on a need-to-know basis. The same reasoning applies also to related work. Following this rationale, the rest of the thesis is organized as follows. In Chapter 2 we present the contributions related to MOMO, in Chapter 3 we present the contributions related to ACE and ACME, and in Chapter 4 we present the contributions related to CryptoAC; a reader interested in a specific contribution may even take different reading paths for Chapters 2 to 4 — although we recommend following the presentation order. We conclude the thesis with final remarks and future work in Chapter 5.

The content of the present thesis is partially based on the following (already peer-reviewed) articles:

- (Chapter 2) 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 Transactions on Privacy and Security, 25, 1, Article 2 (February 2022), 37 pages. DOI: 10.1145/3474056;
- (Chapter 4) Stefano Berlato, Roberto Carbone, and Silvio Ranise. 2021. *Cryptographic Enforcement of Access Control Policies in the Cloud: Implementation and Experimental Assessment*. In Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT 2021), SciTePress. DOI: 10.5220/0010608003700381;
- (Chapter 4) Stefano Berlato, Roberto Carbone, Umberto Morelli, and Silvio Ranise. 2022. *End-to-End Protection of IoT Communications Through Cryptographic Enforcement of Access Control Policies*. In Proceedings of the 36th Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSec 2022), Springer Link. DOI: 10.1007/978-3-031-10684-2_14;
- (Chapter 2) Stefano Berlato, Silvio Cretti, Domenico Siracusa, Silvio Ranise. 2024. *Multi-Objective Microservice Orchestration: Balancing Security and Performance in CCAM*. In Proceedings of the 27th Conference on Innovation in Clouds, Internet and Networks (ICIN 2024), IEEE.

The content of the present thesis is partially based on the following (still under review) articles:

- (Chapter 3) Stefano Berlato, Roberto Carbone, and Silvio Ranise. 2024. *A Methodology for the Experimental Evaluation of Access Control Enforcement Mechanisms based on Business Processes*. Under review at Elsevier Journal of Information Security and Applications (JISA);
- (Chapters 2 and 4) Stefano Berlato, Roberto Carbone, Umberto Morelli, and Silvio Ranise. 2024. *Optimizing Security and Quality of Service for End-to-End Cryptographic Access Control in IoT Applications*. Under review at Elsevier Internet of Things (IoT).

# Chapter 2

# Optimal Placement of Microservices for Cryptographic Access Control

Researchers proposed several CAC schemes relying on a variety of cryptographic primitives and techniques, such as traditional Public Key Infrastructure (PKI) [135, 11], Identity-Based Encryption (IBE) [39], Attribute-Based Encryption (ABE) [115, 59], lazy revocation [133], Proxy Re-Encryption (PRE) [99], and onion encryption [95]. Despite their differences, the vast majority of these CAC schemes are built upon three sets of components, namely assets, entities, and domains. To the best of our knowledge — as also argued in Section 4.7 — prior research has not addressed the *problem* of selecting the optimal assignment of entities to domains (that we call *placement*) for CAC schemes that considers both the performance requirements and the trust assumptions of a particular scenario. On the contrary, the design of many CAC schemes expects a fixed assignment of entities to domains (i.e., a fixed placement), resulting in a lack of flexibility and adaptability to different scenarios.

Therefore, below we formalize the aforementioned *problem* in a MOCOP and demonstrate its usefulness in finding the optimal placement for CAC schemes in 2 different scenarios for cloud native applications, i.e., remote patient monitoring and Cooperative Connected and Automated Mobility (CCAM). First, after providing the necessary background notions (Section 2.1), we define an architectural model describing the assets, entities, and domains that typically compose a CAC scheme (Section 2.2); the definition of the architectural model is fundamental to enumerate all possible placements for CAC schemes — that is, the space of the possible solutions to the MOCOP. Then, we identify a set of objectives derived from a heterogeneous set of performance requirements and trust assumptions relevant to cloud native applications; given a placement, we show how to quantify the satisfaction of these objectives through suitably-defined objective functions (Section 2.3). Afterward, we reduce the problem of selecting the placement maximizing the values of these objective functions to a decidable MOCOP for which we provide a conceptual application on the remote patient monitoring scenario (Section 2.4). Finally, we implement an algorithm solving the MOCOP in `MOMO` and provide a proof-of-concept application in the CCAM scenario by showing how `MOMO` can be directly integrated as a placement algorithm for K8s (Section 2.5). In fact, as said in Section 1.5, the concepts of entities and domains — in the context of cloud native applications — map to the concepts of microservices and computing regions, respectively. We report an overview of the architectural model and the MOCOP in Figure 2.1, and collect symbols used throughout the chapter in Table 2.1.

## 2.1 Background and Preliminaries on Cryptographic Access Control

As mentioned in Section 1.4, AC requests are granted or denied according to an AC policy, which contains rules stating what users can perform what actions on what resources. The AC policy is

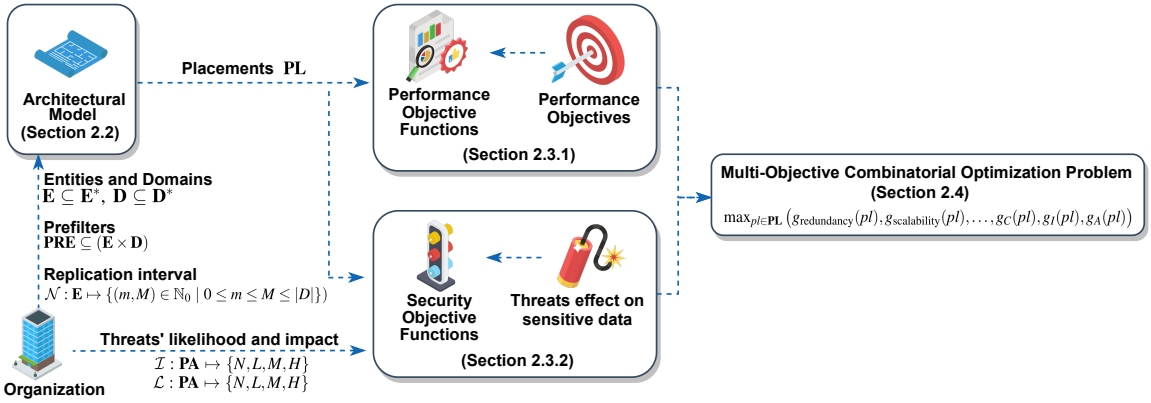


Figure 2.1: Overview of the architectural model and the MOCOP

defined by an administrator, which usually corresponds to the owner of the resources or the system (e.g., the organization). AC enforcement mechanisms for AC policies may be either traditional (i.e., relying on a trusted central entity) — which, however, may not always be suitable for cloud native applications, as argued in Section 1.4 — or based on CAC [8]. Typically, CAC employs both symmetric and asymmetric cryptography to encrypt resources and distribute the corresponding

Table 2.1: Symbols used in Chapter 2

| Symbol | Description | Symbol | Description |
|---|---|---|---|
| $\mathbf{D}^*$ | The set of domains | $\mathbf{E}^*$ | The set of entities |
| $\mathbf{A}$ | The set of assets | $a$ | An asset |
| $\mathbf{E}$ | The subset of considered entities | $e$ | An entity |
| $\mathbf{D}$ | The subset of considered domains | $d$ | A domain |
| $pl$ | A placement (entities-domains assignment) | $\mathbf{PL}^*$ | The set of placements |
| $\mathbf{PL}$ | The set of considered placements | $\mathbf{PL}_{opt}$ | The set of Pareto Optimal placements |
| $\mathbf{M}$ | The set of containers MOMO considers | $m$ | A container |
| $\mathbf{R}$ | The set of computing regions MOMO considers | $r$ | A computing region |
| $nr$ | The node of the computing region $r$ | $DF$ | The data flow considered by MOMO |
| $\mathbf{O}$ | The set of objectives | $o$ | An objective |
| $o_P$ | The set of performance objectives | $\mathbf{O}_P$ | A performance objective |
| $o_S$ | The set of security objectives | $\mathbf{O}_S$ | A security objective |
| $g_o$ | The objective function for the objective $o$ | $w_o$ | The weight of the objective $o$ |
| $\mathbf{PA}$ | The set of points of attack | $pa$ | A point of attack |
| $\mathbf{C}$ | The set of communication channels | $c$ | A communication channel |
| $\mathbf{T}$ | The set of targets | $t$ | A target |
| $\mathbf{IF}$ | The set of information flows | $if$ | An information flow |
| $\mathbf{SP}$ | The set of CIA security properties | $sp$ | A security property |
| $\mathcal{P}$ | The power set | + | Positive influence of an assignment on $o_P$ |
| - | Negative influence of an assignment on $o_P$ | = | Negligible influence of an assignment on $o_P$ |
| $\mathbf{PRE}$ | The set of pre-filters | $\mathcal{L}$ | The likelihood function for threats |
| $\mathcal{I}$ | The impact function for threats | $\mathcal{F}$ | Mapping targets $\mapsto$ assets security properties |
| $\mathcal{D}$ | Mapping assets security properties $\mapsto$ data security properties | $\mathcal{S}$ | Mapping providing a unified view of Equations (2.2) to (2.4) |
| $\mathcal{M}$ | Mapping of the risk table in Table 2.5 | $\mathcal{G}$ | Security objective function |
| $\mathcal{N}$ | Replication interval for entities | $\mathcal{I}_{nf}$ | Entity-domain assignment influence on $o_P$ |
| $u$ | A user | $f$ | A resource |
| $\mathbf{k}^{\mathbf{enc}}$ | A public encryption key | $\mathbf{k}^{\mathbf{dec}}$ | A private decryption key |
| $\mathbf{k}^{\mathbf{sym}}$ | A symmetric key | $al$ | An assurance level |

cryptographic decrypting keys to authorized users, respectively. The use of both symmetric and asymmetric cryptography is usually called *hybrid cryptography* [39]. Authenticated Encryption with Associated Data (AEAD) [6] (e.g., AES-GCM, xsalsa20-poly1305) is usually used for symmetric cryptography, while different cryptographic primitives and techniques (e.g., PKI, IBE, ABE, PRE) may be used for asymmetric cryptography.

As an example, consider the presence of a resource $f$ (e.g., a file) and a user $u$ equipped with a pair of (asymmetric) public-private keys $(\mathbf{k}_u^{\mathbf{enc}}, \mathbf{k}_u^{\mathbf{dec}})$ for encryption and decryption, respectively. Now, assume that the administrator wants to grant $u$ read/write access over $f$; to abstract from low-level details, we assume that all communications occur through pairwise-authenticated and private channels (e.g., with TLS) and that users operate independently of each other with devices (e.g., laptop) protected with suitable means (e.g., antivirus, passwords for access). With this setup, the administrator would first generate a symmetric key $\mathbf{k}_f^{\mathbf{sym}}$ to encrypt the (potentially large quantities of) data contained in $f$, resulting in $\{f\}_{\mathbf{k}_f^{\mathbf{sym}}}$. Then, the administrator would encrypt $\mathbf{k}_f^{\mathbf{sym}}$ with $\mathbf{k}_u^{\mathbf{enc}}$, resulting in $\{\mathbf{k}_f^{\mathbf{sym}}\}_{\mathbf{k}_u^{\mathbf{enc}}}$. Finally, the administrator would send to $u$ (or allow $u$ to access, e.g., by downloading them from a cloud storage service) both $\{f\}_{\mathbf{k}_f^{\mathbf{sym}}}$ and $\{\mathbf{k}_f^{\mathbf{sym}}\}_{\mathbf{k}_u^{\mathbf{enc}}}$. To read the encrypted data, $u$ would decrypt $\{\mathbf{k}_f^{\mathbf{sym}}\}_{\mathbf{k}_u^{\mathbf{enc}}}$ with $\mathbf{k}_u^{\mathbf{dec}}$, obtaining $\mathbf{k}_f^{\mathbf{sym}}$, and then decrypt $\{f\}_{\mathbf{k}_f^{\mathbf{sym}}}$ with $\mathbf{k}_f^{\mathbf{sym}}$. To write on $f$ (e.g., to add new content to the file), $u$ would simply use $\mathbf{k}_f^{\mathbf{sym}}$ to encrypt the updated $f$. Summarizing, in CAC the AC policy is encoded by the encrypted cryptographic keys (i.e., $\{\mathbf{k}_f^{\mathbf{sym}}\}_{\mathbf{k}_u^{\mathbf{enc}}}$) into a CAC policy. Moreover, depending on the specific CAC scheme, the CAC policy may be enriched with auxiliary information (e.g., version numbers, digital signatures).

## 2.2 Architectural Model

As introduced in Section 1.5 and mentioned at the beginning of Chapter 2, CAC schemes are based on three sets of components: assets (Section 2.2.1), entities (Section 2.2.2), and domains (Section 2.2.3). Shown in Figure 2.2, our architectural model identifies and describes these components, allowing for enumerating all possible placements for CAC schemes (Section 2.2.4).

### 2.2.1 Assets

The set of assets **A** comprises the following 4 categories of information related to CAC:

- **resources**: abstract vessels — such as documents, files, communication channels, and message queues — containing (often sensitive or personal) data. As said in Section 2.1, each resource $f$ is typically associated with a (usually distinct) symmetric key $\mathbf{k}_f^{\mathbf{sym}}$;
- **encrypted resources**: resources whose data are encrypted with the corresponding symmetric keys according to the CAC policy;
- **secret keys**: private and secret cryptographic keys — such as the decryption key $\mathbf{k}_u^{\mathbf{dec}}$ of a user $u$ and the symmetric key $\mathbf{k}_f^{\mathbf{sym}}$ associated with a resource $f$;
- **metadata**: further auxiliary information necessary to the functioning of the CAC scheme (e.g., the CAC policy, public cryptographic keys, version numbers, digital signatures).

Resources and secret keys must be kept confidential, that is, known by authorized users only, while encrypted resources are not confidential — if we assume perfect encryption (i.e., the confidentiality and integrity of data protected with cryptography cannot be compromised by attacking the available cryptographic primitives). Finally, metadata may be confidential or not depending on the underlying scenario and the organization's judgment. For example, file names may allow inferring what projects

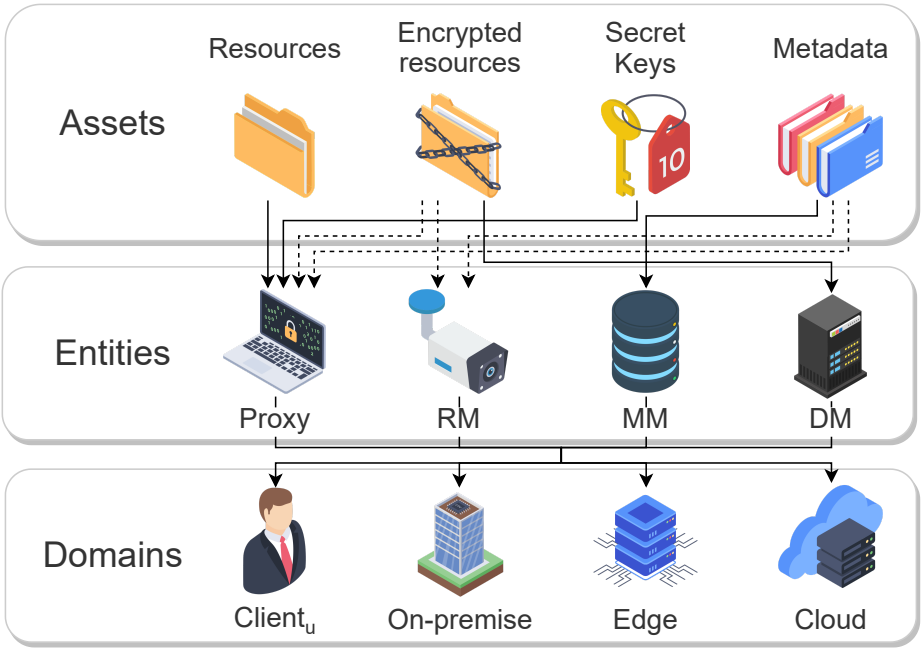


Figure 2.2: Architectural model for CAC schemes — a solid line between an asset and an entity indicates that the entity *contains* the asset, while a dashed line between an asset and an entity indicates that the entity *employs* the asset (see Section 2.2.2)

an organization is working on, while the CAC policy may reveal its internal hierarchy. The integrity of assets is guaranteed through MACs and digital signatures, usually created by the administrator.

### 2.2.2 Entities

The set of entities $\mathbf{E}^*$ comprises the following 4 active components, where each component *contains* or *employs* assets to carry out some part of a CAC scheme:

- **proxy**: this entity — fundamental in CAC [32] — performs cryptographic computations, allowing users to interface with data and the administrator to manage the CAC policy. In other words, the proxy translates high-level AC requests (e.g., read a resource) into the sequence of low-level cryptographic computations necessary to accomplish them (e.g., obtain the cryptographic decryption key, download and decrypt the encrypted resource to obtain the plaintext data). Intuitively, the proxy *contains* secret keys and resources, which must exist — at least in plaintext form — within the proxy only. Moreover, the proxy *employs* metadata and encrypted resources to obtain and use symmetric keys, as explained in Section 2.1;

- **Reference Monitor (RM)**: cryptography alone cannot prevent malicious users from tampering with resources, e.g., by overwriting their content with spurious data. For this reason, many CAC schemes expect the presence of a RM ensuring that users' AC requests to add and write over resources are authentic (e.g., by verifying digital signatures) and compliant with the CAC policy [39]. Note that some CAC schemes do not require a RM, either because resources cannot be overwritten but only appended (e.g., as in [11]) or because resources are versioned; in the latter case, after an AC request to overwrite a resource, that resource is not replaced but a new version of the resource is added and expected to be validated by the next user accessing the resource (e.g., as in [133, 36]). In any case, the RM does not *contain* any asset, but just *employs* metadata and encrypted resources to ensure the compliance of users' AC requests with the CAC policy;

- **Metadata Manager (MM)**: as said in Section 2.2.1, besides secret keys, CAC schemes expect further auxiliary information, referred to as metadata, which needs to be securely managed by a MM — usually, a database or a similar means. Intuitively, the MM does not *employ* any assets, but just *contains* metadata;

- **Data Manager (DM)**: encrypted resources are shared among users by either traveling through the network (i.e., data in transit) or being stored in a repository (i.e., data at rest). In both cases, the DM is the entity managing the encrypted resources — usually, a storage or messaging service (e.g., a broker mediating communications among IoT devices). The functionalities typically offered by a DM follow the Create, Read, Update and Delete (CRUD) paradigm. Intuitively, the DM does not *employ* any assets, but just *contains* encrypted resources.

We highlight that CAC schemes — and the corresponding CAC enforcement mechanisms — might allow for multiple instances of the same entity. For instance, the proxy may exist as a single instance (e.g., as a server hosted within the organization) or as multiple instances (e.g., as software installed in each user's personal device), or even both. Consequently, each instance of an entity would access the corresponding subset of assets (e.g., an instance of a proxy installed in a user $u$'s laptop would access $u$'s secret keys only).

### 2.2.3 Domains

The set of domains $\mathbf{D}^*$ contains the following 4 logical or physical locations that are operated by a participating party and can host one or more entities:

- **client**$_u$: the domain where a user $u$ operates is defined as the set of $u$'s personal devices (e.g., laptop and smartphone). As said in Section 2.1, each user operates independently from other users;

- **on-premise**: this domain corresponds to a secure area within the organization that can be operated by authorized personnel only (e.g., a data center to which only authorized personnel can access, either physically or virtually);

- **edge**: this domain corresponds to computing, network, and storage services geographically located at the outskirts of the network and operated typically by public cloud providers;[1,2]

- **cloud**: this (logical and geographically distributed) domain corresponds to computing, network, and storage services operated by a cloud provider (e.g., AWS, GCP, Azure).

As discussed in Section 1.3, the threat model for cloud native applications assumes minimal trust on the participating parties (e.g., users, cloud providers). Therefore, the client$_u$ domain is typically regarded as untrusted, whereas the edge and the cloud domains as partially trusted, and the on-premise domain as trusted — although the amount of trust may vary depending on the scenario.

### 2.2.4 Assembling the Architectural Model

To summarize the discussion in Sections 2.2.2 to 2.2.4, the sets $\mathbf{A} = \{$resources, encrypted resources, secret keys, metadata$\}$, $\mathbf{E}^* = \{$proxy, RM, MM, DM, app$\}$, and $\mathbf{D}^* = \{$client$_u$, on-premise, edge, cloud$\}$ constitute our architectural model for CAC schemes, as shown in Figure 2.2. By considering all possible assignments of entities to domains, which we denote as the set $\mathbf{PL}^* = \mathcal{P}(\mathbf{E}^* \times \mathbf{D}^*)$ —

[1] IoT Edge, Open Source Edge - AWS IoT Greengrass (https://aws.amazon.com/greengrass/)
[2] Google Distributed Cloud (https://cloud.google.com/distributed-cloud)

where $\mathcal{P}$ is the symbol for the power set— we note that there exist ($|\mathbf{PL}^*|$ =) 65,536 (i.e., $2^{(4\cdot4)}$) different possible placements for CAC schemes — where $|\cdot|$ denotes the cardinality of a set. As shown in Figure 2.1, an organization can modify the architectural model and further tailor the space of the possible solutions to its scenario by specifying:

- the subset of entities $\mathbf{E} \subseteq \mathbf{E}^*$ and domains $\mathbf{D} \subseteq \mathbf{D}^*$ to consider in their scenario. For instance, if a CAC scheme does not expect the presence of the RM, then RM $\notin \mathbf{E}$;
- the set of pre-filters $\mathbf{PRE} \subseteq (\mathbf{E} \times \mathbf{D})$ containing entity-domain assignments not compatible with the threat model considered by the organization. For instance, expecting the RM to be hosted in the client$_u$ domain would allow users to bypass the RM and tamper with data (e.g., unauthorized overwrite). Similarly, expecting the cloud to host the proxy would ideally give the cloud provider the possibility of accessing the secret keys contained within the proxy and, consequently, decrypting the encrypted resources, resulting in the violation of the confidentiality of the data. Nonetheless, this possibility may be negated in scenarios employing technologies offering confidential computing such as Trusted Execution Environments (TEEs)[3] which — thanks to their availability in cloud services[4,5] — were already used in the design of some CAC schemes (e.g., [30]);
- the replication interval $\mathcal{N} : \mathbf{E} \mapsto \{(m,M) \in \mathbb{N}_0 \mid 0 \leq m \leq M \leq |D|\})$ defining the minimum ($m$) and maximum ($M$) number of domains an entity can be assigned to in the same placement. Indeed, depending on the scenario, different domains can host (distinct instances of) the same entity at the same time. For instance, as mentioned in Section 2.2.1, if an organization deems a portion of the metadata to be sensitive (e.g., resource names), it may prefer to keep these metadata in a MM hosted on-premise, while other non-sensitive metadata (e.g., public keys) in a MM hosted in the cloud — at the cost of handling synchronization and update issues.

The set of placements respecting the aforementioned constraints — that is, the space of the possible solutions — is therefore:

$$\mathbf{PL} = \{pl \in \mathcal{P}(\mathbf{E} \times \mathbf{D}) \mid (pl \cap \mathbf{PRE} = \emptyset) \wedge (\mathcal{N}(e)[0] \leq |\{(e_i,-) \in pl \mid e_i = e\}| \leq \mathcal{N}(e)[1]\ \forall (e,-) \in pl)\}; \quad (2.1)$$

## 2.3 Objectives

As said in Chapter 1, each scenario may express specific performance requirements and trust assumptions (e.g., on the threat model) that should be considered when operating and integrating CAC enforcement mechanism — like `CryptoAC` — into cloud native applications. For instance, assigning the MM to the cloud domain may favor redundancy and reliability with respect to assigning the MM to the on-premise domain; however, such an assignment might expose sensible metadata to the cloud provider, compromising their confidentiality (if deemed relevant by the organization). However, if the presence of disgruntled employees within an organization is considered likely and potentially impactful, assigning the MM to the on-premise domain may result in the violation of integrity and availability of metadata. Simultaneously evaluating both performance requirements and trust assumptions is far from being trivial, especially when conflicts arise and it is necessary to conduct a *what-if* analysis to explore the trade-offs among the various placements in **PL** — see Equation (2.1).

[3] E.g., ARM Trustzone (https://developer.arm.com/ip-products/security-ip/trustzone)

[4] Confidential Computing | Google Cloud (https://cloud.google.com/confidential-computing)

[5] Azure Confidential Computing – Protect Data In Use (https://azure.microsoft.com/en-us/solutions/confidential-compute/)

To enable organizations to explore these trade-offs, we identify performance requirements and trust assumptions relevant to cloud native applications and correspondingly derive a set of objectives $\mathbf{O}$ considering both performance objectives $\mathbf{O}_P$ and security objectives $\mathbf{O}_S$ (hence, $\mathbf{O} = \mathbf{O}_P \cup \mathbf{O}_S$). Note that the set of objectives $\mathbf{O}$ is not meant to be exhaustive or representative of all scenarios, and further objectives, e.g., relevant to specific cloud native applications, may be easily added. Then, we show how to evaluate placements against a set of objective functions, where an objective function $g_o : \mathbf{PL} \mapsto \mathbb{Z}$ measures (with an integer value) how much a placement attains an objective $o \in \mathbf{O}$; the internal definition of $g_o$ (i.e., how to compute its value) depends on the corresponding objective $o$, as we discuss in Sections 2.3.1 and 2.3.2.

### 2.3.1 Performance Objectives

By surveying works from both academia and industry,[6] we identify 14 performance requirements that may be desirable in cloud native applications, and map them in as many performance objectives. To evaluate the influence of a placement $pl \in \mathbf{PL}$ on these performance objectives (see Figure 2.1), we adopt a modular approach by considering how each entity-domain assignment $(e,d) \in pl$ affects each performance objective $o_P \in \mathbf{O}_P$ in isolation; more precisely, we establish that an assignment may have a positive (+), negligible (=) or negative (-) influence on the satisfaction of a performance objective; we determine such influences by analyzing the aforementioned surveyed works, and argue our findings in the bulleted list below. As an example, hosting the proxy in the $\text{client}_u$ domain has a positive influence (i.e., +) on scalability — as cryptographic computations are distributed among users — while hosting the proxy on-premise has a negative influence (i.e., -) on scalability — as the proxy would likely become a performance bottleneck for the whole cloud native application. Finally, to measure how much a placement $pl \in \mathbf{PL}$ affects a performance objective $o_P \in \mathbf{O}_P$ — that is, to define the corresponding performance objective function $g_{o_P}$ — we combine the influences of all entity-domain assignments of $pl$ by adding together the +, - and = influences as adding the +1, -1 and 0 integers, respectively. If we define as $\mathcal{I}_{nf} : (\mathbf{E} \times \mathbf{D} \times \mathbf{O}_P) \mapsto \{+1, -1, 0\}$ the function that returns the influence of an assignment $(e,d)$ on a performance objective $o_P$, then we can define the performance objective function for the performance objective $o_P \in \mathbf{O}_P$ as $g_{o_P}(pl) = \sum_{(e,d) \in pl} \mathcal{I}_{nf}(e,d,o_P)$. Below, we report the 14 performance objectives and discuss the influence of entity-domain assignments (which we also summarize in Table 2.2); after each performance objective, we report between round brackets those works that discuss its relevance in cloud native applications:

- **redundancy** [68, 112, 58, 132, 29]: this performance objective represents the extent to which a placement allows for duplicating assets effectively. By achieving geographical distribution and offering consolidated mechanisms for replicating assets, the cloud greatly enhances redundancy [29]. Conversely, the on-premise domain is limited to one or a few physical locations, and the edge relies on nodes — possibly already in use by other tenants — to act as redundant servers, resulting in a limited redundancy capacity [132];

- **scalability** [109, 115, 133, 68, 64, 94, 29, 132, 63]: this performance objective represents the ability of a placement to scale up and down to accommodate dynamic workloads (e.g., a variable number of users' AC requests). Similar to redundancy, scalability is a peculiarity of cloud computing [29] as well as the edge [63, 90]. However, the most scalable placement is, intuitively, the one that distributes the burden of cryptographic computations among users (i.e., by assigning multiple instances of the proxy to the $\text{client}_u$ domain);

[6] E.g., Gigaom Key Criteria for Evaluating File-Based Cloud Storage (https://gigaom.com/report/gigaom-radar-for-file-based-cloud-storage)

- **reliability** [96, 109, 115, 68, 64, 58]: this performance objective represents the resilience of a placement to, e.g., the failure of (a portion of) the computing infrastructure of a domain. While the cloud is reliable by definition [29], the same cannot be said for the edge — which has a higher chance of failure for nodes and a lower fault tolerance [63, 90] — and the on-premise domain — which, instead, is inherently a Single Point of Failure (SPOF);
- **maintenance** [109, 68, 112, 58, 94]: this performance objective represents the easiness in the maintenance of a placement (in terms of, e.g., setup, configuration, and update of the underlying computing infrastructure, operative systems, and runtime environments). Intuitively, relying on domains operated by third parties (i.e., the cloud and the edge) delegates the maintenance issue to the third parties themselves, while hosting entities in the on-premise domain leads to a greater maintenance effort by the organization;
- **DoS resilience** [39, 96, 68, 64, 22]: this performance objective represents the resilience of the placement to DoS attacks. The cloud is intrinsically resilient to DoS attacks [29], while the edge may be more easily affected by even modest DoS attempts — but not as easily as the on-premise domain [22, 90];
- **bandwidth** [90, 14, 63, 132, 29]: this performance objective represents the maximum throughput over (a given path of) the network. The cloud is characterized by high bandwidth [29], while the edge, typically located at the outskirts of the network, generally has lower bandwidth — although still considerable, especially in comparison to the on-premise domain [14, 90];
- **latency** [114, 14, 90, 132, 63]: this performance objective represents the delay in communicating a bit of data. The edge is well-known for its low latency in comparison to the Cloud [90, 132], which is usually even less accessible than on-premise computing infrastructures [114];
- **computational power** [132, 63, 90]: this performance objective represents the ability to execute software and process incoming data. Both the edge and the cloud offer significant computational power, although specific edge nodes — similar to the on-premise domain — might encounter more pronounced constraints on computational power [63, 90];
- **memory** [132, 63, 90]: this performance objective represents the amount of primary memory (i.e., RAM). Similarly to the computational power performance objective, the cloud is more resourceful than the edge and the on-premise domains [63, 132];
- **minimization of edge vendor lock-in** [64, 115]: this performance objective represents the easiness in switching edge provider (e.g., from Google Distributed Cloud to AWS Greengrass). The fewer entities are hosted in the edge domain, the more this performance objective is achieved;
- **minimization of cloud vendor lock-in** [115, 64]: this performance objective represents the easiness in switching cloud provider (e.g., from GCP to AWS). As for the edge, the fewer entities are hosted in the cloud domain, the more this performance objective is achieved;
- **on-premise monetary savings** [115, 68, 64, 58, 94]: this performance objective represents the monetary savings related to the usage of the on-premise domain. The fewer entities are hosted in the on-premise domain, the more this performance objective is achieved;
- **edge monetary savings** [68, 90, 115, 63]: this performance objective represents the monetary savings related to the usage of the edge domain. The fewer entities are hosted in the edge domain, the more this performance objective is achieved;

- **cloud monetary savings** [115, 68, 64, 58, 94]: this performance objective represents the monetary savings related to the usage of the cloud domain. The fewer entities are hosted in the cloud domain, the more this performance objective is achieved.

As one may expect, Table 2.2 shows that assignments including the cloud domain yield positive influences on several performance objectives. Furthermore, placements expecting assignments of the same entity in multiple domains tend to balance their influence. For instance, hosting the MM in the cloud results in a negative influence on the cloud monetary savings performance objective. However, splitting the metadata between two instances of the MM — hosted in the cloud and on-premise domains, respectively — reduces the amount of information stored and the number of requests sent toward the MM in the cloud, hence lowering cloud-related monetary costs as well; this is reasonable if we consider that cloud providers usually charge organizations on a pay-as-you-go basis.[7] Of course, the influences and the performance objective functions we formulated, being extracted from generic scenarios considered in the literature, are coarse-grained and defined at a high level — so as to preserve general applicability — and cover a wide variety of (but not all possible) scenarios. In other words, we acknowledge that the influences and the performance objective functions may not fully reflect the nuances of distinct scenarios and placements (suffice to say that different cloud providers have different pricing plans for their services). Nonetheless, we remark that our contributions are related to the approach to the optimization of placements for CAC schemes rather than to the definition of the single influences of entity-domain assignments on performance objectives. Indeed, if needed, organizations can easily fine-tune such influences based on their scenarios. Besides, organizations can propose additional performance objectives and even ad-hoc performance objective functions, as we later show in Section 2.5.

[7] Amazon S3 Simple Storage Service Pricing (https://aws.amazon.com/s3/pricing/)

Table 2.2: Influence ($\mathcal{I}_{nf}$) of entity-domain assignments on performance objectives

| Goals | proxy | | | | RM | | | | MM | | | | DM | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $client_u$ | on-premise | edge | cloud | $client_u$ | on-premise | edge | cloud | $client_u$ | on-premise | edge | cloud | $client_u$ | on-premise | edge | cloud |
| redundancy | = | - | - | + | = | - | - | + | = | - | - | + | = | - | - | + |
| scalability | + | - | = | = | + | - | = | = | + | - | = | = | + | - | = | = |
| reliability | = | - | - | + | = | - | - | + | = | - | - | + | = | - | - | + |
| maintenance | = | - | + | + | = | - | + | + | = | - | + | + | = | - | + | + |
| DoS resilience | + | - | = | + | + | - | = | + | + | - | = | + | + | - | = | + |
| bandwidth | - | - | = | + | - | - | = | + | - | - | = | + | - | - | = | + |
| latency | = | = | + | - | = | = | + | - | = | = | + | - | = | = | + | - |
| computational power | - | - | = | + | - | - | = | + | - | - | = | + | - | - | = | + |
| memory | - | - | - | + | - | - | - | + | - | - | - | + | - | - | - | + |
| min. edge vendor lock-in | + | + | - | + | + | + | - | + | + | + | - | + | + | + | - | + |
| min. cloud vendor lock-in | + | + | + | - | + | + | + | - | + | + | + | - | + | + | + | - |
| on-premise monetary savings | + | - | + | + | + | - | + | + | + | - | + | + | + | - | + | + |
| edge monetary savings | + | + | - | + | + | + | - | + | + | + | - | + | + | + | - | + |
| cloud monetary savings | + | + | + | - | + | + | + | - | + | + | + | - | + | + | + | - |

### 2.3.2 Security Objectives

Even though CAC provides E2E protection and selective sharing of sensitive data, a scenario may include specific threats worth considering when selecting the optimal placement for a CAC scheme — as already highlighted in Section 1.3. For instance, consider a scenario involving a remote patient monitoring software application in the eHealth field, as exemplified in the analysis presented in [136]. In such a scenario, IoT wearables may be susceptible to Man-at-the-Device (MatD) attackers, particularly when unattended or during (the confusion arising in the course of) a medical emergency. Furthermore, partially trusted cloud providers may collude with malicious insiders (e.g., disgruntled employees) to gain unauthorized access to patients' data. Additionally, MitM attackers may tamper with communication channels, e.g., those involving the IoT wearables themselves (as said in Section 1.3).

Therefore, to enable organizations to consider these threats when selecting the placement for a CAC scheme, we identify 3 security objectives denoting the level of assurance on the confidentiality ($C$), integrity ($I$), and availability ($A$) of sensitive data, respectively — hence, $\{C,I,A\} = \mathbf{O}_S$. To evaluate the influence of a placement $pl \in \mathbf{PL}$ on these security objectives (see Figure 2.1), we formalize a risk assessment allowing to measure the risk exposure of data — in terms of $C,I$, and $A$ — to these threats in $pl$. For instance, if an organization deems the insider threat to be one of the most impactful attack vectors for its scenario, then a placement assigning all entities to the on-premise domain would expose data to high levels of risk — and may therefore not be the best placement for such a scenario. As typically done in the literature [42], we follow a matrix-based mapping where the risk of a threat (e.g., a malicious insider colluding with a cloud provider) is decomposed into 2 dimensions, i.e., likelihood and impact: the former measures the probability that a specific threat occurs, while the latter measures the potential impact (e.g., damage) of the threat; both likelihood and impact consider 4 different values, i.e., $N$(egligible), $L$(ow), $M$(edium), and $H$(igh). Concerning the likelihood, $N$ and $H$ imply that the chance of a threat occurring are, for all practical aspects, zero and one, respectively, whereas $L$ and $M$ represent probability values in the intermediate range. Concerning the impact, $N$ and $H$ imply that the damage of a threat is negligible and catastrophic, respectively, whereas $L$ and $M$ represent probability values in the intermediate range. The matrix in Table 2.5 shows how to compute risk levels — on the same scale with increasing values from $N$ to $H$ — given the likelihood and the impact of a threat. To preserve the generality of the optimization problem — and not overly focus on a specific scenario — we cluster specific threats (e.g., device cloning, eavesdropping, session hijacking, side-channel attacks, spoofing, spear phishing) into a unique generic threat for each domain and communication channel between pairs of domains. As intuitive, the likelihood and impact of each threat mainly depend on the trust assumptions of the underlying scenario (e.g., see Section 2.4) and must therefore be evaluated by the organization on a case-by-case basis. More generally, likelihood and impact depend on the fact that an exploitable vulnerability exists (e.g., in a software application due to poor implementation or bad configuration, within an organization due to employees not educated against phishing attacks or lack of control on the access to the data center room) and that the vulnerability will be exploited (e.g., by an external attacker or a malicious insider) — indeed, vulnerabilities may be more or less dangerous or difficult to exploit because of mitigations put in place (e.g., anti-malware, employees' training). Nonetheless, although assuming the likelihood and the impact of threats to be provided as input by the organization, we need to determine the effect that these threats would have on the confidentiality, integrity, and availability of sensitive data — and this effect depends on what assets among those described in Section 2.2.1 a threat may (potentially) strike. For instance, an attacker striking the proxy could access the secret keys contained therein (potentially affecting the confidentiality of data), whereas an attacker striking the DM could tamper with encrypted resources (potentially affecting the availability of data).

Below, we first discuss how to determine the effect of threats on sensitive data protected by CAC schemes, and then formalize the risk assessment accordingly.

**Determine Threats Effect on Sensitive Data.** To determine the effect that threats may have on the confidentiality, integrity, and availability — three security properties commonly referred to as "CIA" which we group under the set **SP** — of the sensitive data protected by a CAC scheme, we first determine where a threat might occur by identifying the set of possible *point of attacks* **PA**. As mentioned at the beginning of Section 2.3.2, we consider one generic threat for each domain and communication channel between pairs of domains. Therefore, in our architectural model (see Figure 2.2), a point of attack $pa \in \mathbf{PA}$ is either a domain $d \in \mathbf{D}$ or a communication channel between pairs of domains $c \in \mathbf{C} = \{\{d_i, d_j\} \mid (d_i, d_j) \in (\mathbf{D} \times \mathbf{D}) \wedge d_i \neq d_j\}$ — hence, $\mathbf{PA} = (\mathbf{D} \cup \mathbf{C})$. Then, as argued in [8], it is reasonable to say — by considering the worst-case scenario — that a threat occurring in a domain affects all entities hosted in that domain (e.g., malware in a laptop potentially affects all software running in that laptop). Similarly, a threat (e.g., a MitM attacker) occurring in a communication channel between two domains affects all information flowing through that communication channel. Following this reasoning, we denote with **T** the set of *targets* that threats can affect. In our architectural model, a target is either an entity $e \in \mathbf{E}$ or an information flow between a pair of entities $if \in \mathbf{IF} = \{\{e_i, e_j\} \mid (e_i, e_j) \in (\mathbf{E} \times \mathbf{E}) \wedge e_i \neq e_j\}$ — hence, $\mathbf{T} = (\mathbf{E} \cup \mathbf{IF})$. Formally, given a placement $pl \in \mathbf{PL}$, we define the set of targets $\mathbf{T}_{affected}$ which a threat occurring in a given $pa \in \mathbf{PA}$ can affect as:

$$\mathbf{T}_{affected} = \begin{cases} \{t \in \mathbf{T} \mid (e,d) \in pl \wedge e = t \wedge d = pa\} & pa \in \mathbf{D} \\ \{\{e_1, e_2\} \in \mathbf{T} \mid (e_1, d_1) \in pl \wedge (e_2, d_2) \in pl \wedge \{d_i, d_j\} = pa\} & pa \in \mathbf{C} \end{cases} \tag{2.2}$$

The definition of the set of affected targets $\mathbf{T}_{affected}$ in Equation (2.2) is fundamental for identifying what assets (i.e., those introduced in Section 2.2.1) a threat might strike. Intuitively, a threat may strike an asset when at rest, in use, or in transit. More in detail, assets at rest are those contained by entities, assets in use are those employed by entities, and assets in transit are those transmitted from containing to employing entities (see the representation of the architectural model in Figure 2.2 and the usage of the "*contain*" and "*employ*" verbs in Section 2.2.2). Indeed, the fact that an entity employs an asset implies an information flow between the entity containing the asset and the entity employing the asset; for instance, given that the RM employs metadata (to ensure the compliance of users' AC requests with the CAC policy), then there should exist an information flow between the RM and the MM; in simpler words, the RM needs to fetch the metadata from the MM. Consequently, a threat that affects an entity compromises the confidentiality, integrity, and availability of assets contained by that entity, and the confidentiality and integrity of assets employed by that entity — but not the availability of employed assets (which, indeed, are still available in the corresponding containing entities); as an example, deleting a downloaded file in a user's laptop does not compromise the availability of the original file. The same reasoning applies to threats that affect an information flow, i.e., threats can compromise the confidentiality and integrity of assets in transit but not their availability. Note that the presence of information flows is conditional on the placement being considered; if two entities (e.g., RM and MM) are hosted in the same domain, eventual information flows between the two entities are implicit within that domain, i.e., assets are not in transit through an (external) network. In this case, the information flow simply does not exist — this aspect is already captured in Equation (2.2). We report the mapping of what security properties of assets a threat may compromise by affecting a target in Table 2.3, and denote the mapping with the function $\mathcal{F} : \mathbf{T} \mapsto (\mathbf{A} \times \mathbf{SP})$. For instance, consider a placement $pl \in \mathbf{PL}|(\text{proxy,on-premise}) \in pl$. In this

example, the on-premise domain is the point of attack, while the proxy entity is the target hosted therein; a threat occurring in the on-premise domain would strike the proxy entity, compromising the confidentiality and the integrity of resources, encrypted resources, secret keys, and metadata, as well as the availability of resources and secret keys (see the first row of Table 2.3). Formally, given a placement $pl \in \mathbf{PL}$, we define the set of asset-security property pairs $\mathbf{ASP}_{struck}$ which a threat occurring in a given $pa \in \mathbf{PA}$ can compromise as:

$$\mathbf{ASP}_{struck} = \{(a, sp) \mid (a, sp) \in \mathcal{F}(t) \wedge t \in \mathbf{T}_{affected}\} \tag{2.3}$$

The definition of the set of struck asset-security property pairs $\mathbf{ASP}_{struck}$ in Equation (2.3) is fundamental for understanding the effect that a threat has on the confidentiality, integrity, and availability of the data protected by a CAC scheme; more in detail:

- **resources**: compromising the confidentiality, integrity, or availability of resources affects the confidentiality, integrity, or availability of sensitive data, respectively;
- **encrypted resources**: compromising the integrity or the availability of encrypted resources affects the availability of sensitive data (but not their integrity, as cryptography makes tampering attempts evident thanks to digital signatures and MACs), while compromising the confidentiality of encrypted resources does not affect sensitive data (if assuming perfect encryption, as said in Section 2.2.1);
- **secret keys**: compromising the integrity or the availability of secret keys affects the availability of sensitive data (as tampered secret keys cannot be used for decryption), while compromising the confidentiality of a secret key affects the confidentiality and integrity of sensitive data. Indeed, a secret key can potentially be used to access (i.e., decrypt) large amounts of sensitive data or also to covertly tamper with sensitive data (e.g., by following the same steps users do to generate digital signatures or MACs as described in Section 2.1);
- **metadata**: compromising the integrity or the availability of metadata affects the availability of sensitive data — as explained in Section 2.1. Compromising the confidentiality of metadata does not affect sensitive data, except if an organization deems metadata to be sensitive data themselves (as said in Section 2.2.4).

We report the mapping of what security properties of sensitive data a threat may compromise for each asset in Table 2.4, and denote the mapping with the function $\mathcal{D} : (\mathbf{A} \times \mathbf{SP}) \mapsto \mathcal{P}(\mathbf{SP})$. Formally, given a placement $pl \in \mathbf{PL}$, we define the set of security properties of sensitive data $\mathbf{SP}_{compromised}$ a threat occurring in a given $pa \in \mathbf{PA}$ can compromise as:

Table 2.3: $\mathcal{F}$ — security properties of assets that a threat may compromise by affecting a target

| | | Security Property | | |
|---|---|---|---|---|
| | | Confidentiality | Integrity | Availability |
| Target | proxy | {resources, encrypted resources, secret keys, metadata} | {resources, encrypted resources, secret keys, metadata} | {resources, secret keys} |
| | MM | {metadata} | {metadata} | {metadata} |
| | *RM* | {encrypted resources, metadata} | {encrypted resources, metadata} | $\emptyset$ |
| | DM | {encrypted resources} | {encrypted resources} | {encrypted resources} |
| | {proxy, MM} | {metadata} | {metadata} | $\emptyset$ |
| | {proxy, DM} | {encrypted resources} | {encrypted resources} | $\emptyset$ |
| | {RM, MM} | {metadata} | {metadata} | $\emptyset$ |
| | {RM, DM} | {encrypted resources} | {encrypted resources} | $\emptyset$ |

$$\mathbf{SP}_{compromised} = \{sp \mid sp \in \mathcal{D}(a, sp') \wedge (a, sp') \in \mathbf{ASP}_{struck}\} \tag{2.4}$$

To help the reader in better understand the discussion above, we provide a unified view of the content of this paragraph — that is, Equations (2.2) to (2.4) — as a function $\mathcal{S} : (\mathbf{PL} \times \mathbf{PA}) \mapsto \mathbf{SP}$ whose pseudocode is reported in Algorithm 1. Essentially, $\mathcal{S}$ returns, for a given placement $pl \in \mathbf{PL}$ and point of attack $pa \in \mathbf{PA}$, what security security properties of data a threat occurring in $pa$ would compromise.

**Risk Assessment.** As explained in Section 2.3.2, the impact and likelihood of each threat depend on the trust assumptions of the underlying scenario. Hence, we assume the organization to provide two functions specifying the likelihood and the impact of a threat occurring in a certain point of attack, i.e., $\mathcal{L} : \mathbf{PA} \mapsto \{N, L, M, H\}$ and $\mathcal{I} : \mathbf{PA} \mapsto \{N, L, M, H\}$, respectively. As mentioned in Section 2.3.2, we can combine likelihoods and impacts following the matrix-based mapping proposed in [42] as reported in Table 2.5 — we denote the matrix-based mapping with the function $\mathcal{M} : (\{N, L, M, H\} \times \{N, L, M, H\}) \mapsto \{N, L, M, H\}$. Finally, we combine the risk assessment with the function $\mathcal{S}$ to determine the risk exposure on the confidentiality, integrity, or availability of data associated with each threat as a function $\mathcal{G} : (\mathbf{PL} \times \mathbf{SP}) \mapsto \mathbb{Z}_{\leq 0}$ whose pseudocode is reported in Algorithm 2. As done in Section 2.3.1, to measure the risk exposure — that is, to define the security objective functions $g_C, g_I$, and $g_A$ — we combine the risk levels due to each threat by summing them, security objective-wise, assigning to the risk levels $N, L, M$, and $H$ the integers 0, 1, 2, and 3, respectively. Since our goal is to maximize the values of objective functions (as said at the beginning of Chapter 2), we make $\mathcal{G}$ returns, for a given placement $pl$ and security property $sp$, an integer whose value is *lower* the higher the risk exposure of $pl$ for the $sp$ of sensitive data. Concretely, instead of summing together the risk levels (converted to integers), we subtract them from an *assurance level* whose initial value is set to 0. In other words, $\mathcal{G}(pl, sp)$ computes the assurance level of the security objective corresponding to $sp$, i.e., $g_C(pl) := \mathcal{G}(pl, \text{confidentiality}), g_I(pl) := \mathcal{G}(pl, \text{integrity})$ and $g_A(pl) := \mathcal{G}(pl, \text{availability})$.

Table 2.4: $\mathcal{D}$ — security properties of data compromised if an asset-security property pair is struck

| Compromised Asset-Security Property Pair | | Security Property of Sensitive Data | | |
|---|---|---|---|---|
| | | Confidentiality | Integrity | Availability |
| resources | confidentiality | × | | |
| | integrity | | × | |
| | availability | | | × |
| encrypted resources | confidentiality | | | |
| | integrity | | | × |
| | availability | | | × |
| secret keys | confidentiality | × | × | |
| | integrity | | | × |
| | availability | | | × |
| metadata | confidentiality | ×* | | |
| | integrity | | | × |
| | availability | | | × |

*if metadata are considered to be sensitive data

## 2.4 Multi-Objective Combinatorial Optimization Problem

After having defined objective functions for performance and security objectives (in Sections 2.3.1 and 2.3.2, respectively), we can now formalize the problem of finding the placement(s) that simultaneously maximize all objective functions as the following MOCOP (see also Figure 2.1):

$$\max_{pl \in \mathbf{PL}} \left( g_{\text{redundancy}}(pl), g_{\text{scalability}}(pl), \ldots, g_C(pl), g_I(pl), g_A(pl) \right) \tag{2.5}$$

The operator max should be carefully defined since, for any non-trivial MOCOP, there may exist multiple solutions (i.e., placements, in our case) that can be considered equally good [65]. Therefore, as commonly done in MOCOPs, we use the *Pareto Dominance* relation as ordering relation for max: given $pl_1, pl_2 \in \mathbf{PL}$, $pl_1$ dominates $pl_2$ iff $pl_1$ produces equal or higher objective function values than $pl_2$, with at least one value strictly higher; formally $pl_1 \succ pl_2 \iff (\forall o \in \mathbf{O}\ g_o(pl_1) \geq g_o(pl_2)) \wedge (\exists o \in \mathbf{O}\ g_o(pl_1) > g_o(pl_2))$. A $pl \in \mathbf{PL}$ is said to be a *Pareto Optimal* solution to the MOCOP in Equation (2.5) when no placement dominates $pl$, i.e., $\nexists pl' \in \mathbf{PL} : pl' \succ pl \wedge pl' \neq pl$. Finally, we

**Algorithm 1:** $\mathcal{S}$ - data security properties compromised per placement and point of attack

**Input:** $pl, pa$ // Input: placement $pl$ and point of attack $pa$
**Output:** $\mathbf{SP}_{compromised}$ // Output: security properties of data compromised
// 1. Identify targets affected in $pa$ - Equation (2.2)
let $\mathbf{T}_{affected} = \emptyset$
**if** $(pa \in \mathbf{D})$ **then**
  // Each entity in the domain is a target
  **foreach** $((e,d) \in \{(e,d) \in pl \mid d = pa\})$ **do**
    add $e$ to $\mathbf{T}_{affected}$
  **end**
**else**
  // Each information flow between the domains is a target
  **foreach** $((e_1,d_1) \in \{(e,d) \in pl \mid d \in pa\})$ **do**
    **foreach** $((e_2,d_2) \in \{(e,d) \in pl \mid d \in pa\})$ **do**
      **if** $\{d_1,d_2\} = pa$ **then**
        add $\{e_1,e_2\}$ to $\mathbf{T}_{affected}$
      **end**
    **end**
  **end**
**end**
// 2. Identify asset-security property pairs struck in $pa$ - Equation (2.3)
let $\mathbf{ASP}_{struck} = \emptyset$
**foreach** $(t \in \mathbf{T}_{affected})$ **do**
  **foreach** $((a,sp) \in \mathcal{F}(t))$ **do**
    add $(a,sp)$ to $\mathbf{ASP}_{struck}$
  **end**
**end**
// 3. Identify data security properties compromised in $pa$ - Equation (2.4)
let $\mathbf{SP}_{compromised} = \emptyset$
**foreach** $((a,sp) \in \mathbf{ASP}_{struck})$ **do**
  add $\mathcal{D}(a,sp)$ to $\mathbf{SP}_{compromised}$
**end**
**return** $\mathbf{SP}_{compromised}$

Table 2.5: Risk table [42]

| Likelihood \ Impact | *N* | *L* | *M* | *H* |
|---|---|---|---|---|
| *N* | *N* | *N* | *N* | *N* |
| *L* | *N* | *L* | *L* | *M* |
| *M* | *N* | *L* | *M* | *H* |
| *H* | *N* | *M* | *H* | *H* |

highlight that the set of possible solutions (i.e., **PL**) to the MOCOP is finite, hence the MOCOP is surely decidable [65]. Moreover, we can identify the set of Pareto Optimal placements $\mathbf{PL}_{opt}$ reusing off-the-shelf the approaches available in the literature to solve MOCOPs. For instance, a straightforward approach consists in treating the MOCOP in Equation (2.5) as a multi-dimensional maximum vector problem and solving it using the `Best` algorithm described in [41] and shown in Algorithm 3. `Best` first iterates over the set of possible solutions **PL** to find a *single* Pareto Optimal solution $pl^*$. Then, `Best` removes from **PL** all solutions that are not Pareto Optimal with respect to $pl^*$. In following iterations, more solutions can be found. Finally, `Best` ends when **PL** is empty.

**Computational Complexity of `Best`.** Let $|\mathbf{O}|$ be the number of objective functions, $|\mathbf{PL}|$ the number of possible solutions (i.e., placements), and $|\mathbf{PL}_{opt}|$ the number of Pareto Optimal solutions. The first step to solve the MOCOP in Equation (2.5) is to evaluate all objective functions for

**Algorithm 2:** $\mathcal{G}$ - Security objective function for the security objective corresponding to *sp*

**Input:** $pl, sp$ `// Input: placement` $pl$ `and security property` $sp$
**Output:** $al_{sp}$ `// Output: assurance level` $al$ `for security property` $sp$
let $al_{sp} = 0$
`// Determine the risk level for each point of attack`
**foreach** $(pa \in \mathbf{PA})$ **do**
  `// If the security property` $sp$ `of data is affected by the threat in` $pa$
  **if** $sp \in \mathcal{S}(pl, pa)$ **then**
    let $rl = \mathcal{M}(\mathcal{L}(pa), \mathcal{I}(pa))$ `// Compute the risk level of the threat`
    let $rli =$ `riskLevelToInteger`$(rl)$ `// Convert the risk level to an integer`
    $al_{sp} = al_{sp} - rli$ `// Substract the risk level as integer from` $al_{sp}$
  **end**
**end**
**return** $al_{sp}$

`// Return the risk level as an integer`
**Function** `riskLevelToInteger`$(rl)$**:**
  let $n = 0$
  **if** $(rl = N)$ **then**
    $n = 0$
  **else if** $(rl = L)$ **then**
    $n = 1$
  **else if** $(rl = M)$ **then**
    $n = 2$
  **else if** $(rl = H)$ **then**
    $n = 3$
  **return** $n$

**Algorithm 3:** `Best` algorithm from [41]

**Input: PL**
**Output:** $\mathbf{PL}_{opt}$ `// Set of Pareto Optimal Solutions`
$\mathbf{PL}_{opt} = \emptyset$
**while** $\mathbf{PL} \neq \emptyset$ **do**
  $pl^* =$ pop(**PL**)
  `// Find a Pareto Optimal solution`
  **foreach** $pl \in \mathbf{PL}$ **do**
    **if** $(pl \succ pl^*)$ **then**
      $pl^* = pl$
  **end**
  $\mathbf{PL}_{opt} = \mathbf{PL}_{opt} \cup \{pl^*\}$
  `// Remove architectures not Pareto Optimal w.r.t` $pl^*$
  **foreach** *pl in* **PL** **do**
    **if** $(pl^* \succ pl)$ **then**
      $\mathbf{PL} = \mathbf{PL} \setminus \{pl\}$
  **end**
**end**
**return** $\mathbf{PL}_{opt}$

each placement; this is easily seen to have complexity $O(|\mathbf{O}| \cdot |\mathbf{PL}|)$. Then, as `Best` iterates over **PL** once for each Pareto Optimal solution, the complexity depends on $|\mathbf{PL}_{opt}|$. In the best case there is a single Pareto Optimal solution (i.e., $|\mathbf{PL}_{opt}| = 1$) and `Best` iterates only twice on **PL** by making $|\mathbf{O}|$ comparisons for each $pl \in \mathbf{PL}$, with a complexity of $O(|\mathbf{O}| \cdot |\mathbf{PL}|)$. The worst case, where every architecture is optimal (i.e., $|\mathbf{PL}_{opt}| = |\mathbf{PL}|$), requires $|\mathbf{PL}|$ iterations with a complexity of $O(|\mathbf{O}| \cdot |\mathbf{PL}|^2)$. Finally, in the average case (i.e., $1 < |\mathbf{PL}_{opt}| < |\mathbf{PL}|$), the complexity is $O(|\mathbf{O}| \cdot |\mathbf{PL}| \cdot |\mathbf{PL}_{opt}|)$. Generally, the complexity of `Best` is the sum of $O(|\mathbf{O}| \cdot |\mathbf{PL}|)$ — to calculate the values of objective functions — and $O(|\mathbf{O}| \cdot |\mathbf{PL}_{opt}| \cdot |\mathbf{PL}|)$ — to compute $\mathbf{PL}_{opt}$. Therefore, the final complexity is $O(|\mathbf{O}| \cdot |\mathbf{PL}| \cdot |\mathbf{PL}_{opt}|)$.

### 2.4.1 Application to the Remote Patient Monitoring Scenario

To demonstrate the usefulness of the MOCOP in finding the optimal placement for CAC schemes, we now provide a conceptual application on the remote patient monitoring scenario: we first describe the scenario, then configure the MOCOP accordingly, and finally discuss the obtained results (i.e., Pareto Optimal placements).

**The Remote Patient Monitoring Scenario.** The remote patient monitoring scenario has been widely studied in the literature [1] by several researchers (e.g., see [32, 51, 73, 85, 109, 133, 94, 99, 136]) either as a whole or by focusing on a specific aspect of the scenario (e.g., data collection by IoT wearables, data storage in the cloud). Indeed, patients' medical data (e.g., blood sugar and LDL cholesterol levels) are extremely delicate and thus require secure management — in terms of transmission, storage, and access. In particular, confidentiality and integrity are fundamental security properties of medical data (due to the need to preserve patients' privacy and the fact that medical data may be used for diagnoses); availability — albeit valued — is not as important [81]. Then, Egala et al. [34] highlight the need to outsource the storage of medical data to avoid SPOFs, enhance redundancy, and avoid data loss. Intuitively, this need has to be balanced with the presence of honest-but-curious cloud providers — that are likely interested in medical data. In this regard, Domingo-Ferrer et al. [32] point out that even metadata (e.g., patient's name, diseases) may leak confidential information, and need therefore to be protected from cloud providers. For instance, suppose a person with a mental disorder is hospitalized in a clinic specialized for treating such a type of disorders. If the patient's name is present in the metadata of the CAC schemes (e.g., in the CAC policy), a cloud provider may then infer that a specific person is a customer of the clinic. Consequently, the cloud provider may share this information for targeted advertisement or with a health insurance company that may then increase the insurance premium of the person [7]. Instead, the presence of disgruntled doctors is generally considered as unlikely, especially when considering private clinics [8].

**Configuring the MOCOP for the Remote Patient Monitoring Scenario.** We consider a CAC scheme with all entities and domains in the architectural model presented in Section 2.2, i.e., $\mathbf{E} = \mathbf{E}^*$, $\mathbf{D} = \mathbf{D}^*$, and $\mathbf{PL} = \mathbf{PL}^*$.

Concerning the pre-filters, the need to outsource the storage of medical data prevents the DM from staying in the on-premise domain. Moreover, we do not consider technologies offering confidential computing for the sake of simplicity (recall the discussion in Section 2.2.4). Hence, the proxy cannot stay in the cloud and edge domains — as, otherwise, the cloud provider could access the secret keys — while the RM cannot stay in the $\text{client}_u$ domain — as, otherwise, users could bypass the RM to tamper with data (e.g., unauthorized overwrite). Finally, as intuitive, we assume that neither the DM nor the MM are hosted in the $\text{client}_u$ domain. Formally: $(\text{DM},\text{client}_u), (\text{MM},\text{client}_u), (\text{DM},\text{on-premise}), (\text{proxy},\text{edge}), (\text{proxy},\text{cloud}), (\text{RM},\text{client}_u) \in \mathbf{PRE}$. Then, concerning the replication interval (see Section 2.2.4), the need to hide metadata from cloud providers may be translated with the

definition of at least 2 replicas for the MM, one of which contains the sensitive part of the metadata and should therefore be assigned to the on-premise domain; replicas for other entities can be left as default. Formally: $\mathcal{N}(\text{MM}) = (2, |\mathbf{D}|)$, and $\mathcal{N}(\text{proxy}) = \mathcal{N}(\text{RM}) = \mathcal{N}(\text{DM}) = (1, |\mathbf{D}|)$. Finally, regarding trust assumptions, the fact that cloud providers may actively try to access medical data sets the likelihood of a threat occurring in the cloud or edge domain to high. Instead, the unlikely presence of malicious insiders sets the likelihood of a threat occurring in the on-premise domain to low. Without additional information, the likelihood for the $\text{client}_u$ domain and all communication channels may be set to a default value, i.e., medium. Finally, the impact is set to high wherever a threat may compromise large quantities of medical data (i.e., cloud, edge, and on-premise domains), to low wherever a threat may compromise medical data of a single patient (i.e., $\text{client}_u$ domain), and to a default value (i.e., medium) for communication channels. Again, we remark that our goal is to provide a conceptual application of the architectural model in Section 2.2 and the MOCOP in Equation (2.5), and not to actually propose a placement for CAC schemes in the remote patient monitoring scenario.

**Pareto Optimal Placements for the Remote Patient Monitoring Scenario.** We implement the performance and security objective functions discussed in Sections 2.3.1 and 2.3.2, respectively, the MOCOP in Equation (2.5) — configured as described in the previous paragraph — and the `Best` algorithm presented in Algorithm 3 into a python script.[8] Running the script — that amounts to solving the MOCOP for the remote patient monitoring scenario — returns 26 Pareto Optimal placements among the 252 placements remaining after considering the pre-filters and the replication interval. We report the 26 Pareto Optimal placements in Table 2.6. Once reduced the search space from 65,536 (recall the definition of $\mathbf{PL}^*$ in Section 2.2.4) to 26 placements, an organization can now choose the placement most suitable to the underlying scenario. This choice may require some ingenuity by experts in security and in the other fields to which the objectives belong (e.g., performance and quality of service), which may evaluate the Pareto Optimal placements independently and then meet to reach a consensus on the placement that strikes the best possible trade-off. Alternatively, we highlight that there exist several ways (e.g., Minimum-Cost Flow [48], Generalized Assignment Optimization Problem [91]) to transform MOCOPs with multiple Pareto Optimal solutions into (more or less) equivalent optimization problems with one — or at least fewer — solutions; one of the simplest ways — that we describe below — is to reduce the MOCOP to a Single-Objective Optimization Problem (SOOP). We refer the interested reader to [75] for a comprehensive survey on the topic of solving MOCOPs.

**Single-Objective Optimization Problem.** Reducing the MOCOP in Equation (2.5) to a SOOP amounts at assigning, to each objective $o \in \mathbf{O}$, a weight $w_o$ representing the relative importance or priority of $o$; in other words, we construct an objective function of the form $\sum_{o \in \mathbf{O}} (w_o \cdot g_o)$. The SOOP then consists in finding the placement for which the value of the linear combination of all weighted objective functions is maximum:

$$\max_{pl \in \mathbf{PL}} \left( w_{\text{redundancy}} \cdot g_{\text{redundancy}}(pl) + \ldots + w_A \cdot g_A(pl) \right) \tag{2.6}$$

When considering a SOOP, the organization can perform a trade-off analysis by fine-tuning the weights, exploring the space of possible solutions. For instance, to identify the placement which maximizes the satisfaction of performance objectives only, an organization could set $w_C = w_I = w_A = 0$. The final choice of which placement to select is then trivial — i.e., one among those for which the value computed from Equation (2.6) is maximum. Note that weights need to be positive

[8] mocop.py (https://drive.google.com/file/d/1J5otA4QRfDUK5ohcwIllVjanLNNjAwXw/view?usp=sharing)

to guarantee that the solutions to the SOOP in Equation (2.6) are also Pareto Optimal solutions for the MOCOP in Equation (2.5) [65]. Finally, the computational complexity of solving the SOOP in Equation (2.6) is simply $O(|\mathbf{PL}| \cdot |\mathbf{O}|)$ (i.e., for each placement, compute and then do the weighted sum of all objective function values).

The description of the remote patient monitoring scenario at the beginning of Section 2.4.1 says that confidentiality and integrity are more important than availability [81], and that redundancy is

Table 2.6: Pareto Optimal architectures for the remote patient monitoring scenario. For each row, the first four columns show the assignments of entities to domains (i.e., the placement), the remaining columns — except the last — show the objective functions values for each objective, and the last column shows the linear combination of all weighted objective functions values according to the discussion at the end of Section 2.4.1. The first two architectures are optimal with respect to the SOOP formulated in Equation (2.6)

| | Domain | | | redundancy | scalability | reliability | maintenance | DoS resilience | bandwidth | latency | computational power | memory | min. edge lock-in | min. cloud lock-in | on-premise savings | edge savings | cloud savings | confidentiality | integrity | availability | linear combination |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| client$_u$ | on-premise | edge | cloud | | | | | | | | | | | | | | | | | | |
| | | | | 1 | -2 | 1 | 1 | 1 | 1 | -3 | 1 | 1 | 5 | -1 | 1 | 5 | -1 | -7 | -2 | -7 | -10 |
| | | | | 2 | 0 | 2 | 2 | 3 | 1 | -3 | 1 | 1 | 5 | -1 | 3 | 5 | -1 | -12 | -1 | -12 | -10 |
| | | | | -5 | -2 | -5 | -1 | -2 | -4 | 2 | -4 | -6 | 2 | 6 | 0 | 2 | 6 | -12 | -3 | -12 | -73 |
| | | | | 0 | -1 | 0 | 0 | 1 | -1 | -2 | -1 | -1 | 5 | 1 | 1 | 5 | 1 | -12 | -1 | -12 | -30 |
| | | | | -5 | -2 | -5 | 1 | -2 | -2 | 3 | -2 | -5 | -1 | 5 | 1 | -1 | 5 | -7 | -2 | -7 | -40 |
| | | | | -3 | -1 | -3 | 1 | 0 | -2 | 1 | -2 | -4 | 2 | 4 | 2 | 2 | 4 | -19 | -3 | -21 | -76 |
| | | | | -2 | 0 | -2 | 2 | 1 | -1 | 1 | -1 | -3 | 1 | 3 | 3 | 1 | 3 | -17 | -1 | -19 | -57 |
| | | | | -3 | -2 | -3 | -1 | -1 | -3 | 0 | -3 | -4 | 4 | 4 | 0 | 4 | 4 | -12 | -3 | -19 | -65 |
| | | | | -1 | -1 | -1 | 1 | 1 | -1 | -1 | -1 | -2 | 4 | 2 | 2 | 4 | 2 | -19 | -3 | -21 | -67 |
| | | | | -1 | -2 | -1 | 1 | 0 | 0 | -1 | 0 | -1 | 3 | 1 | 1 | 3 | 1 | -14 | -2 | -14 | -46 |
| | | | | -3 | -3 | -3 | -1 | -2 | -2 | 0 | -2 | -3 | 3 | 3 | -1 | 3 | 3 | -7 | -2 | -12 | -50 |
| | | | | -5 | -3 | -5 | -1 | -3 | -3 | 2 | -3 | -5 | 1 | 5 | -1 | 1 | 5 | -7 | -2 | -7 | -60 |
| | | | | 0 | 0 | 0 | 2 | 2 | 0 | -1 | 0 | -1 | 3 | 1 | 3 | 3 | 1 | -17 | -1 | -19 | -42 |
| | | | | -4 | 0 | -4 | 2 | 0 | -2 | 3 | -2 | -5 | -1 | 5 | 3 | -1 | 5 | -12 | -1 | -12 | -55 |
| | | | | -3 | -2 | -3 | 1 | -1 | -1 | 1 | -1 | -3 | 1 | 3 | 1 | 1 | 3 | -14 | -2 | -14 | -61 |
| | | | | -3 | -3 | -3 | -1 | -2 | -2 | 0 | -2 | -3 | 3 | 3 | -1 | 3 | 3 | -7 | -2 | -12 | -50 |
| | | | | -3 | -2 | -3 | -1 | -1 | -3 | 0 | -3 | -4 | 4 | 4 | 0 | 4 | 4 | -12 | -3 | -19 | -65 |
| | | | | -4 | -1 | -4 | 0 | -1 | -3 | 2 | -3 | -5 | 1 | 5 | 1 | 1 | 5 | -12 | -1 | -12 | -60 |
| | | | | -2 | -1 | -2 | 0 | 0 | -2 | 0 | -2 | -3 | 3 | 3 | 1 | 3 | 3 | -12 | -1 | -17 | -50 |
| | | | | -1 | -3 | -1 | -1 | -1 | -1 | -2 | -1 | -1 | 5 | 1 | -1 | 5 | 1 | -7 | -2 | -7 | -30 |
| | | | | -5 | -1 | -5 | 1 | -1 | -3 | 3 | -3 | -6 | 0 | 6 | 2 | 0 | 6 | -12 | -3 | -12 | -68 |
| | | | | -3 | -2 | -3 | 1 | -1 | -1 | 1 | -1 | -3 | 1 | 3 | 1 | 1 | 3 | -14 | -2 | -14 | -61 |
| | | | | -2 | -1 | -2 | 0 | 0 | -2 | 0 | -2 | -3 | 3 | 3 | 1 | 3 | 3 | -12 | -1 | -17 | -50 |
| | | | | 1 | -1 | 1 | 1 | 2 | 0 | -3 | 0 | 0 | 6 | 0 | 2 | 6 | 0 | -12 | -3 | -12 | -31 |
| | | | | -1 | -2 | -1 | -1 | 0 | -2 | -2 | -2 | -2 | 6 | 2 | 0 | 6 | 2 | -12 | -3 | -12 | -43 |
| | | | | -1 | -2 | -1 | 1 | 0 | 0 | -1 | 0 | -1 | 3 | 1 | 1 | 3 | 1 | -14 | -2 | -14 | -46 |

a crucial performance objective [34]. Therefore, an organization could assign higher weights to the confidentiality and integrity security objectives (e.g., $w_C = w_I = 2$ while $w_A = 1$) and to the redundancy performance objective (e.g., $w_{\text{redundancy}} = 5$ while $w_{o_P} = 1 \; \forall o_P \in \mathbf{O}_P \setminus \{\text{redundancy}\}$). By doing so, we can now compute the value of the linear combination of all weighted objective functions for each placement, which we report in the last column of Table 2.6. As we can see from the table, the first two placements attain the highest value (i.e., -10). In both cases, the sensitive part of the metadata is contained in the instance of the MM hosted in the on-premise domain. At the same time, to enhance redundancy, encrypted resources are contained in the DM hosted in the cloud domain. The edge domain is not considered, as its characteristics (e.g., low latency — see Table 2.2) are not relevant to the remote patient monitoring scenario.

## 2.5 MOMO

The conceptual application on the remote patient monitoring scenario described in Section 2.4.1 showcases how the MOCOP in Equation (2.5) can assist organizations in identifying optimal placements for CAC schemes — at the theoretical level. However, we claim that our MOCOP can also be directly integrated as a security mechanism within the Orchestration and Management layer of the cloud native stack (see Figure 1.1) during the Operate phase of the DevOps lifecycle (see Figure 1.2) to identify the optimal placement of the microservices composing a CAC scheme into computing regions. To support this claim, we now show the implementation of the MOCOP into MOMO — short for *Multi-Objective Microservice Orchestration* — and provide a proof-of-concept application in a service relevant to the CCAM scenario, i.e., the Cooperative Lane Change (CLC) service.

### 2.5.1 The Cooperative Lane Change Service

CCAM allows autonomous and smart vehicles to communicate with each other to enable a wide array of real-time services and make informed collective driving decisions in concert with other traffic participants with the ultimate goal of increasing efficiency and safety in road transports. The 5G-CARMEN H2020 European project[9] — to which we collaborated — investigated CCAM by proposing several cooperative maneuvering services. In particular, 5G-CARMEN proposed a CLC service in which vehicles install a front-end (FE) enabling the exchange of their position, speed, and intended trajectory through dedicated Cooperative Awareness Messages (CAMs); a message broker mediates the exchange, grouping CAMs by areas of interest (e.g., highway acceleration lanes). Hence, FEs can build an internal representation of nearby traffic conditions (i.e., road awareness) to optimize their driving decisions and make lane change maneuvers easier and safer, either automatically — for autonomous vehicles — or by recommending the best course of action to human drivers (e.g., slow down, do not merge). CLC considers using CAC to guarantee the confidentiality and the integrity of CAMs while allowing authorized vehicles only to participate [24] based on, e.g., their location or capabilities [57].

We report in Figure 2.3 an example of how the CLC service functions when considering two vehicles (i.e., vehicle-A, vehicle-B): FE-A — that is, the FE installed in vehicle-A — sends a CAM to the proxy-A which encrypts the CAM, forwarding the encrypted CAM to the DM and the corresponding symmetric key to the MM (encrypted according to the CAC policy). Then, the proxy-B receives the encrypted CAM from the DM, decrypts it with the corresponding symmetric key fetched from the MM (as described in Section 2.1), and sends the (plaintext) CAM to vehicle-B. As CAMs cannot be modified (i.e., overwritten), the RM is not present in the CAC schemes expected

[9] 5G Carmen (https://5g-ppp.eu/5g-carmen/)

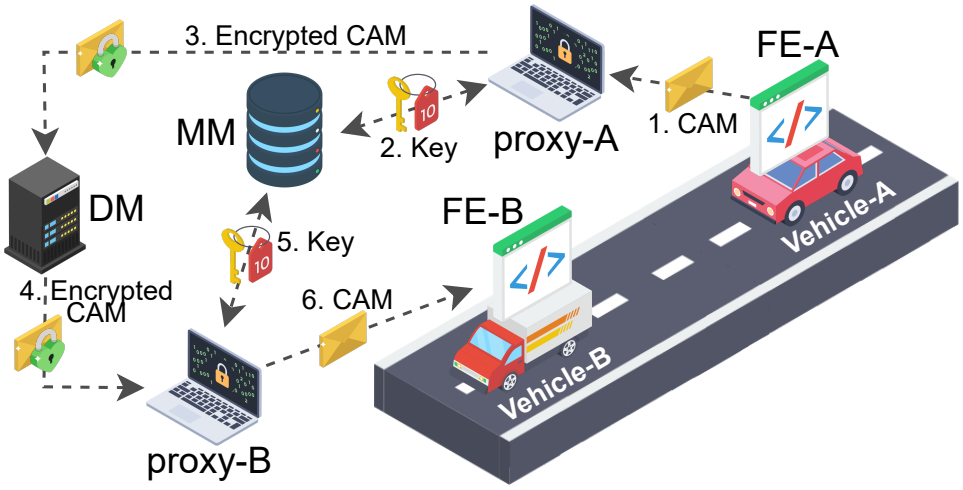


Figure 2.3: Data flow in the cooperative lane change service

by the CLC service (recall the discussion in Section 2.2.2). In CLC, the proxy, the DM, and the MM are implemented as microservices managed by K8s — chosen for its popularity and features such as resiliency, load balancing, and zero-touch deployment — across 4 types of computing regions: cloud, on-premise, edge, and vehicles.[10]

The definition of the placement of microservices into nodes of computing regions is not trivial when considering the peculiarities of CLC. Indeed, like many CCAM services [10, 21], CLC is characterized by a delicate trade-off between performance and security [22]. On the one hand, CLC requires E2E vehicle-to-vehicle latency to be $\leq$ 100ms for the exchange of up-to-date CAMs [24] — bandwidth is not a concern given the limited size of CAMs ($<$ 300 bytes). Similarly, CLC microservices require a certain amount of computational resources to ensure the timely derivation of driving decisions. On the other hand, the placement of the microservices must consider the presence of a heterogeneous set of threats to the confidentiality, integrity, and availability of CAMs [22].

### 2.5.2 `MOMO` for the Cooperative Lane Change Service

As shown in Figure 2.3, the CLC service considers a CAC scheme with all domains and all entities but the RM, i.e., $\mathbf{E} = \mathbf{E}^* \backslash \text{RM}$ and $\mathbf{D} = \mathbf{D}^*$. Then, as mentioned in Section 1.5, in `MOMO` we map the high-level concepts of entity and domain to the concepts of microservices and computing regions; consequently, $\mathbf{E}$ and $\mathbf{D}$ correspond to the set of containers $\mathbf{M}$ and computing regions $\mathbf{R}$, respectively. In this regard, we note that the client$_u$ domain as a computing region is present twice in $\mathbf{R}$, once for each vehicle (i.e., vehicle-A and vehicle-B). Also, we assume that each region $r \in \mathbf{R}$ has only one node $nr$, i.e., a single (physical or virtual) machine on which microservices can run. For the sake of simplicity, we configure the pre-filters just to prevent the MM and the DM from being assigned to the vehicles; formally: $(\text{MM,vehicle-A}), (\text{MM,vehicle-B}), (\text{DM,vehicle-A}), (\text{DM,vehicle-B}) \in \mathbf{PRE}$. Moreover, we set the replication interval to consider one instance of DM and MM and at most two instances of the proxy (i.e., one for each vehicle); formally: $\mathcal{N}(\text{proxy}) = (1,2)$, and $\mathcal{N}(\text{MM}) = \mathcal{N}(\text{DM}) = (1,1)$.

The relevant security objectives are confidentiality ($C$), integrity ($I$) and availability ($A$) of CAMs. Hence, $\mathbf{O}_S = \{C, I, A\}$). Following the arguments in [22], we set the likelihood and impact of threats to high in the cloud and edge domains, and to low for other domains. Concerning communications channels, the authors in [22] highlight that the network to which vehicles are connected is open and

[10] Deploying microservices of cloud native applications in smart vehicles is realistic and believed to be an integral part of the evolution of intelligent transportation systems in both academia [78, 82, 125] and the industry — see https://www.redhat.com/en/blog/running-containers-cars

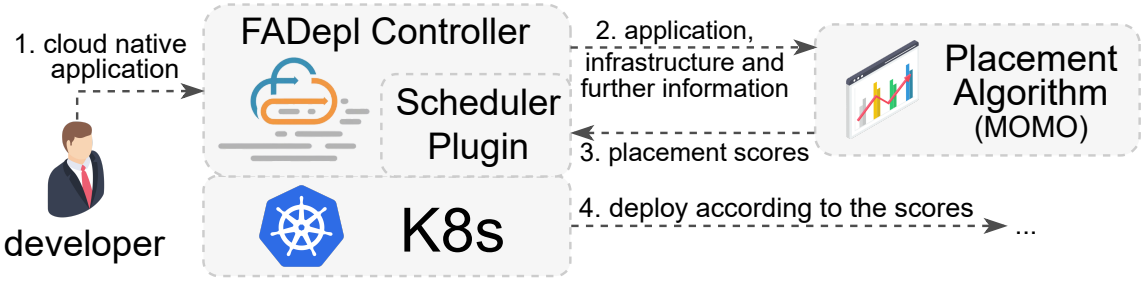


Figure 2.4: Architecture of FogAtlas

public. Therefore, we assign a high value to the likelihood and impact of all communication channels involving the vehicles. Without additional information, we set a default value (i.e., medium) for the likelihood and the impact of the remaining communication channels.

As we can infer from the description of the CLC service in Section 2.5.1, there are four relevant performance requirements which are related to computational resources (i.e., computational power and memory), bandwidth and (E2E) latency; for brevity, we refer to these performance requirements as $CPU, MEM, BAN$, and $LAT$, respectively, and map them in as many performance objectives. Hence, $\mathbf{O_P} = \{CPU, MEM, BAN, LAT\}$. However, at this point, one may notice how in CLC we need a more fine-grained approach to assess the satisfaction of the performance objectives than the default performance objective function presented in Section 2.3.1. For this reason, we propose alternative ad-hoc performance objective functions and report them in Table 2.7: in brief, a placement must be feasible according to the available computational resources ($g_{CPU}, g_{MEM}$) while E2E Latency ($g_{LAT}$) considers cumulative latency across the service data flow (i.e., the one in Figure 2.3) — network bandwidth ($g_{BAN}$) is not a concern in CLC. Instead, we can reuse the security objective functions $g_C$, $g_I$, and $g_A$ defined in Section 2.3.2.

Finally, since deriving driving decisions timely is crucial for safety, we seek the most secure among the placements that satisfy all performance objectives. Hence, we can solve the MOCOP using a *bounded objective function method* [75] that maximizes a tuple of objective functions — ($g_c$, $g_I$, $g_A$), in our case — and uses others (i.e., $g_{CPU}, g_{MEM}, g_{BAN}$ and $g_{LAT}$) to form additional constraints; formally, $\max_{pl \in \mathbf{PL}} (g_C(pl), g_I(pl), g_A(pl))$ *subject to* $0 < g_{o_P}(pl) \leq 1 \; \forall o_P \in \mathbf{O_P}$.

**Integration of `MOMO` with FogAtlas.** Figure 2.4 shows the architecture of FogAtlas: (1) a cloud native application, encoded as a graph of microservices and modeled as a K8s Custom Resource Definition with a FADepl resource,[11] is sent to the FADepl Controller component; this (2) invokes

[11] fogatlas-k8s crd-client-go (https://gitlab.fbk.eu/fogatlas-k8s/crd-client-go)

Table 2.7: Definition of performance objective functions for the cooperative lane change service*

| | |
|---|---|
| $g_{CPU}(pl) =$ | 1 if $(\sum_{(m,r) \in pl} m_{CPU}) \leq nr_{CPU} \; \forall r \in \mathbf{R}$, else 0 |
| $g_{MEM}(pl) =$ | 1 if $(\sum_{(m,r) \in pl} m_{MEM}) \leq nr_{MEM} \; \forall r \in \mathbf{R}$, else 0 |
| $g_{BAN}(pl) =$ | 1 (network bandwidth is not a concern in CLC [24]) |
| $g_{LAT}(pl) =$ | 1 if $(\sum_{(m_1,m_2) \in DF} LAT(r_1, r_2)) \leq$100ms — where $(m_1, r_1)$, $(m_2, r_2) \in pl$ — else 0 |

*$m_{CPU}$ and $m_{MEM}$ are the processing and memory computational requirements of a microservice $m$, while $nr_{CPU}$ and $nr_{MEM}$ are the processing and memory computational resources available in the node $nr$ in the region $r$. $DF$ is the data flow shown in Figure 2.3, i.e., $DF = \{$ (FE-A, proxy-A), (proxy-A, MM), (proxy-A, DM), (DM, proxy-B), (proxy-B, MM), (MM, proxy-B), (proxy-B, FE-B) $\}$, while $LAT(r_1, r_2)$ is the latency between $r_1$ and $r_2$ (see Figure 2.5).

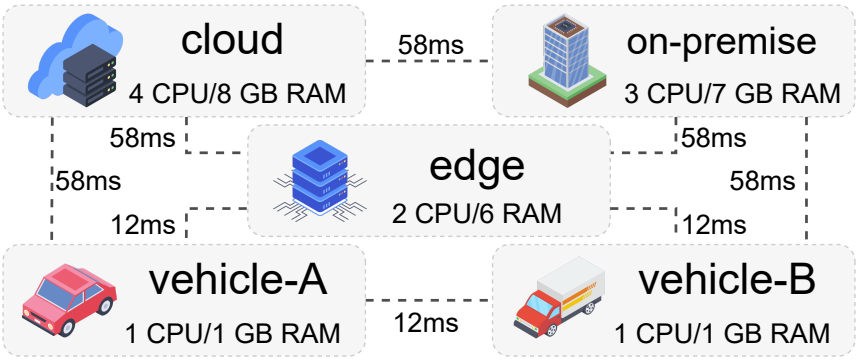


Figure 2.5: K8s cluster demo setup

a Placement Algorithm (i.e., `MOMO`) with, as input, the cloud native application (i.e., **M**), the infrastructure (i.e., **R**), and further information to compute objective functions (i.e., computational resources available in each node, network resources, computational resources required by each microservice, E2E latency constraint). The Placement Algorithm (3) computes Placement Scores encoding the best placement identified; the Scheduler Plugin (a custom plugin implemented in the K8s Scheduling Framework acting during the *score phase* of the K8s scheduling cycle) uses the scores to assign a node to each microservice. Finally, K8s (5) deploys the microservices on the infrastructure accordingly.

### 2.5.3 Application to the Cooperative Lane Change Service

To apply `MOMO` to the CLC service, we configure an emulated environment with a K8s cluster (ver. 1.25) deployed in the data center of the *Fondazione Bruno Kessler* research center.[12] As in Figure 2.5, the cluster has the 5 regions presented in Section 2.5.2. We refer to experimental settings validated in 5G-CARMEN [25] and the work in [89] for *distributing* the cluster with realistic network latency and computational resources for each node. For the purpose of the demo, we reduce the computational resources by a factor of 8 to fit the cluster in our data center and tune the latency with the `tc` utility. As for the microservices implementing the entities, we choose the (open-source and popular) Redis data store as MM, the Mosquitto MQTT broker as DM, and `CryptoAC` — which implements CAC for MQTT — as proxy (see Chapter 4). Since 5G-CARMEN lacks the implementation for the FE (and driving decisions are not relevant in our demo), we implement FE-A/B using `Locust`[13] and `ACME` (see Chapter 3) to measure the E2E latency of a CAM going from vehicle-A to vehicle-B (see Figure 2.3). Finally, we derive computational resource requirements of the MM, DM, and the proxy from the corresponding documentation. For consistency with the cluster, we scale requirements on computational resources by 8 and report them in Table 2.8 (the CPU is in milli-units, the MEM in MB).[14]

[12] Fondazione Bruno Kessler (https://www.fbk.eu/)
[13] Locust (https://locust.io/)
[14] The replication package is available at gitlab.fbk.eu/fogatlas-k8s/uc-ccam

Table 2.8: Requirements of the microservices of the cooperative lane change service

| Microservice | CPU | MEM | Microservice | CPU | MEM |
|---|---|---|---|---|---|
| proxy | 130 | 64 | MM | 50 | 50 |
| DM | 50 | 50 | | | |

**Experiments.** We now demonstrate the effectiveness of MOMO in identifying the placement that satisfies the performance requirements while maximizing the security of CAMs in CLC. To this end, we compare MOMO with 3 other placement strategies, for a total of 4 experiments:

1. **K8s** - as a baseline strategy, we use the default placement of K8s, which just assigns microservices to regions equipped with more computational resources;
2. **performance** - we use MOMO only for the satisfaction of performance constraints only ($0 < g_{o_P}(pl) \leq 1 \; \forall o_P \in \mathbf{O}_P$);
3. **security**: we use MOMO only for the maximization of the security of CAMs only ($\max_{pl \in \mathbf{PL}} (g_C(pl), g_I(pl), g_A(pl))$);
4. MOMO: we use MOMO as described in Section 2.5.2, i.e., $\max_{pl \in \mathbf{PL}} (g_C(pl), g_I(pl), g_A(pl))$ *subject to* $0 < g_{o_P}(pl) \leq 1 \; \forall o_P \in \mathbf{O}_P$.

In all 4 experiments, we run the corresponding Placement Algorithm, deploy the CLC microservices accordingly (note that FE-A/B are already assigned to vehicle-A/B, respectively), and use Locust and ACME to measure the E2E latency of the transmission of 360 CAMs — each experiment lasts 6 minutes so to reduce measurement errors — computing the E2E latency mean and median.

**Results.** We report the placements, security objective function values, and median/mean E2E latency results in Table 2.9 (we omit constraints on computational resources as all placements respect them). Note that E2E latency includes also the microservices processing time (around 15ms) which is relatively small (4%—37%) but not irrelevant. The *K8s* and *security* experiments do not respect the E2E latency constraint ($\leq$ 100ms), as the former just assigns microservices to the most resourceful nodes ignoring latency among regions, while the latter achieves the highest *possible* security, assigning the proxy to vehicles-A/B — for CAMs E2E protection — and the MM and DM to the cloud, the most secure region (but also distant from vehicles). The *performance* experiment fulfills all constraints by relying on the edge. However, $g_C, g_I$, and $g_A$ have lower values: one of the reasons is that CAMs sent from vehicle-A are protected by the proxy only after reaching the edge, and are thus exposed to the threats present in the (public) network in-between. MOMO, on the one hand, improves E2E latency while losing little security with respect to the *security* experiment and, on the other hand, improves security while entailing slightly higher (but still acceptable) E2E latency with respect to the *performance* experiment. In other words, the placement proposed by MOMO is not the best either performance- or security-wise, but it allows to effectively balance a heterogeneous set of relevant — and seemingly conflicting — objectives.

Table 2.9: Experiments placements, security, and E2E latency

| **Experiment** | **Microservices and Regions*** | | | | **Security**** | | | **E2E Latency** | |
|---|---|---|---|---|---|---|---|---|---|
| | **proxy-A** | **MM** | **DM** | **proxy-B** | $g_C$ | $g_I$ | $g_A$ | **Median** | **Mean** |
| *K8s* | cloud | on-premise | edge | on-premise | $-12$ | $-7$ | $-7$ | 373ms | 374ms |
| *performance* | edge | edge | edge | edge | $-12$ | $-7$ | $-7$ | **40ms** | **40ms** |
| *security* | vehicle-A | on-premise | on-premise | vehicle-B | $-2$ | $-7$ | $-2$ | 373ms | 375ms |
| MOMO | vehicle-A | edge | edge | vehicle-B | $-2$ | $-7$ | $-7$ | **94ms** | **95ms** |

# Chapter 3

# Realistic Performance Evaluation for Access Control Enforcement Mechanisms

CAC provides E2E protection and privileged access management for sensitive data in cloud native applications. However, while enhancing data security, CAC might also incur non-negligible computational overhead, as mentioned at the beginning of Chapter 1. More importantly, this computational overhead likely varies according to (the complexity and dynamicity of) the underlying scenario. For this reason, we believe that organizations may want to evaluate such a computational overhead by measuring the performance of a CAC enforcement mechanism as accurately and realistically as possible (with respect to the underlying scenario) *beforehand* — that is, before deploying the mechanism into their cloud native applications. Moreover, organizations may also want to compare the performance of a CAC enforcement mechanism with those of other (traditional and centralized) mechanisms, with the final goal of making an educated choice on what AC enforcement mechanism to use in their cloud native applications — by considering the security guarantees inherent to different mechanisms in light of their performance (recall the discussion about finding the optimal trade-off between performance and security in Chapter 1). Intuitively, besides performance (and security), an organization may compare different mechanisms also based on *subjective* conditions such as open-source license terms, commercial monetary costs, and ease of integration with existing software; however, subjective conditions are hardly measurable by definition, and are thus out of the scope of our work.

To the best of our knowledge, prior research addressed the *problem* of evaluating the performance of AC enforcement mechanisms by focusing on their functionalities, i.e., from the point of view of the developers of the mechanisms, through the use of micro-benchmarks. Essentially, micro-benchmarks consist in making AC enforcement mechanisms evaluate single AC requests (e.g., read a resource) repeatedly, often while ranging, e.g., the number of roles (e.g., [9]) or attributes (e.g., [102, 76]) in the AC policy. Unfortunately, micro-benchmarks produce performance results which — although definitively useful — are intrinsically generic, hence they cannot truly capture what the performance of an AC enforcement mechanism would be under workloads representative of the underlying scenario. In other words, we believe that micro-benchmarks are not representative of a realistic use of AC enforcement mechanisms, as they cannot assess the overall user experience in the use of the application, i.e., from the users' point of view. Instead, we believe that a more realistic approach would be to evaluate the performance of a mechanism under a specific sequence of AC requests derived from a realistic (business) process. Moreover, this sequence should include both users' (e.g., access a resource) and administrative (e.g., assign a permission) AC requests. The rationale is that organizations use AC enforcement mechanisms to evaluate users' and administrative AC requests while performing a specific business process. For instance, the AC request of a warehouse worker

to read the store inventory database (e.g., to check the availability of a product) could be part of a (much larger) order fulfillment process which includes several other activities (e.g., remove an article from the catalog, create an order procurement AC request, send an invoice to the employees of the billing office, update the catalog with a new article). Therefore, while a micro-benchmark would evaluate the performance of an AC enforcement mechanism in relation to a single AC request at a time (e.g., corresponding to the store inventory read activity alone), a more comprehensive evaluation would consider the whole order fulfillment process. Even more realistically, the performance of a mechanism should be evaluated through the concurrent and intertwined execution of more instances of different business processes of the same organization (e.g., order fulfillment, shipment, payment) to simulate a representative scenario. Generalizing, the usage of an AC enforcement mechanism — and, consequently, its performance — largely depends on the business processes of the underlying scenario defining the kind and number of AC requests sent to the mechanism; unfortunately, micro-benchmarking cannot capture this fundamental aspect, as we further discuss in Sections 3.2.3 and 3.5.4. While it is reasonable that no mechanism is better than the others in all scenarios — and that performance should be carefully evaluated on a case-by-case basis — no alternative to micro-benchmarking is available to evaluate the performance of AC enforcement mechanisms.

Therefore, below we address the aforementioned *problem* by proposing a two-module methodology providing the means to measure the performance — in terms of scalability, response time, and throughput — of AC enforcement mechanisms through the simulation of realistic scenarios. To this end, we base our methodology on BPMN workflows, that allow for an appropriate abstraction of the sequences of AC requests (e.g., access a resource, revoke a permission) sent toward AC enforcement mechanisms in a given scenario. First, we provide the necessary background notions (Section 3.1). Then, we give an overview of our two-module methodology (Section 3.2): the first module — implemented by `ACE` — allows for deriving realistic sequences of AC requests (e.g., revoke a permission, access a resource) from BPMN workflows automatically (Section 3.3), while the second module — implemented by `ACME` — allows for evaluating and comparing the performance of AC enforcement mechanisms using the sequences of AC requests previously derived by `ACE` (Section 3.4). We provide a proof-of-concept application of `ACE` and `ACME` by using them to evaluate the performance of three AC enforcement mechanisms representative of both traditional centralized — i.e., the OPA and the XACML — and decentralized AC — i.e., `CryptoAC`;[1] we also compare the results of our methodology against those obtained from micro-benchmarks, confirming our claim that the latter may not always suggest the best mechanism for a given scenario (Section 3.5). Finally, we compare our methodology with related work (Section 3.6). We collect symbols used throughout the chapter in Table 3.1.

## 3.1 Background and Preliminaries for our Methodology

Below, we provide background information on BPMN and WF nets (Sections 3.1.1 and 3.1.2). Then, we introduce relevant AC concepts and present state blueprints — later used in our evaluation — derived from real-world datasets (Section 3.1.3).

### 3.1.1 Business Process Model and Notation

BPMN is an Object Management Group (OMG) standard[2] to model business processes intuitively through BPMN workflows (or simply "workflows") readily understandable by all stakeholders (e.g.,

[1] XACML is an OASIS standard comprising a language, a reference architecture and a dedicated protocol to express and evaluate AC policies based on an extension of the ABAC model, while OPA is an open-source AC enforcement mechanism designed for cloud native applications that provides a high-level declarative language (called Rego) to specify AC policies

[2] About the Business Process Model and Notation Specification Version 2.0.2 (https://www.omg.org/spec/BPMN/)

managers, developers). BPMN uses a common set of symbols meticulously described in the standard [47]; the unambiguousness of these symbols and their independence of any particular implementation environment enable organizations to employ workflows developed in different scenarios to achieve the same business goals. Similarly to programming patterns, workflows can be reused to implement recurring business processes (e.g., project administration, HR management), reducing costs and improving efficiency. Consequently, organizations often reuse consolidated and well-established BPMN workflows instead of developing new ones from scratch. For instance, the ITSM Process Library is a commercial collection of BPMN workflows "for sustainable and value-creating process-oriented IT operations".[3]

**BPMN Symbols.** A workflow represents a business process through (several instances of) 3 kinds of *flow objects*:

- **events**: represented as circles, events are triggers that start (circle with thin line), occur during (circle with double line), or end (circle with thick line) a workflow;

[3]ITSM Process Library | Signavio (https://www.signavio.com/reference-models/itsm-process-library/)

Table 3.1: Symbols used in Chapter 3

| Symbol | Description | Symbol | Description |
|---|---|---|---|
| $\langle P,T,F\rangle$ | A WF net | $P$ | The set of places of a WF net |
| $T$ | The set of transitions of a WF net | $F$ | The set of edges of a WF net |
| $t$ | A transition | $p$ | A place |
| $p_{source}$ | The (unique) source place of a WF net | $p_{sink}$ | The (unique) sink place of a WF net |
| ${}^\bullet t$ | The pre-set of the transition $t$ | $t^\bullet$ | The post-set of the transition $t$ |
| $m(p)$ | The marking of the place $p$ in WF net | $\mathbb{N}$ | The set of natural numbers |
| $m[t\rangle$ | Denoting that the transition $t$ is enabled in the marking $m$ | $[m\rangle$ | The set of markings recursively derivable from the marking $m$ |
| $m_0$ | The canonical marking of a WF net | $m_s$ | The ending marking of a WF net |
| $X$ | The set of (complete) WF net executions | $x$ | A WF net execution |
| $\langle \Gamma, Q, \vdash, \mathbf{\Psi}\rangle$ | An AC scheme | $\Gamma$ | The set of states |
| $Q$ | The set of queries | $\mathbf{\Psi}$ | The set of state-change rules |
| $\vdash$ | The entailment relation | $\langle \gamma, \Psi\rangle$ | An AC system |
| $\gamma$ | A state | $\Psi$ | A subset of the set of state-change rules |
| $\langle \gamma, \psi, ..., \gamma'\rangle$ | An AC execution | $\psi$ | A state-change rule |
| $\gamma \mapsto_\psi \gamma'$ | The state-change rule $\psi$ transforms $\gamma$ in $\gamma'$ | $\mathbf{U}$ | The set of users |
| $\mathbf{R}$ | The set of roles | $\mathbf{F}$ | The set of resources |
| $\mathbf{UR}$ | The set of user-role assignments | $\mathbf{PA}$ | The set of role-permission assignments |
| $\mathbf{PR}$ | The set of permissions | $\mathbf{OP}$ | The set of actions |
| $q$ | A query | $u$ | A user |
| $r$ | A role | $p$ | A permission |
| $f$ | A resource | $op$ | An action |
| $m_{(u\to r)}$ | The min number of user-role assignments | $M_{(u\to r)}$ | The max number of user-role assignments |
| $m_{(r\to u)}$ | The min number of role-user assignments | $M_{(r\to u)}$ | The max number of role-user assignments |
| $m_{(p\to r)}$ | The min number of permission-role assignments | $M_{(p\to r)}$ | The max number of permission-role assignments |
| $m_{(r\to p)}$ | The min number of role-permission assignments | $M_{(r\to p)}$ | The max number of role-permission assignments |
| $\mathcal{S}$ | The function mapping a workflow to a WF net | $\mathcal{D}$ | The function decorating WF net executions |
| $\mathcal{E}$ | The function identifying WF net executions in a WF net | $\mathcal{A}$ | The function deriving list of sequences of state-change rules from decorated WF net executions |
| $W$ | A workflow | $o$ | A topological order |
| $v$ | A vertex | $s$ | A BPMN symbol |
| $l$ | A BPMN pool or lane | $\cdot$ | A wildcard value |
| $req_t$ | The set of data incoming to transition $t$ | $prod_t$ | The set of data outgoing from transition $t$ |
| $x^d$ | A decorated WF net execution | $X^d$ | The set of decorated WF net executions |
| $c$ | A sequence of state-change rules | $C$ | A list of sequences of state-change rules |

- **activities**: represented as (usually rounded) rectangles, activities are tasks to be performed during the execution of the workflow;
- **gateways**: represented as diamonds, gateways are decision points defining conditional behavior to regulate the flow of activities and events. A gateway can be, e.g., exclusive ("x"), parallel ("+"), inclusive ("o") or event-based ("⬠").

Two flow objects can then be linked together with *connecting objects*, the most common of which are:

- **sequences**: represented with solid arrows, sequence connecting objects express the order in which activities are performed;
- **messages**: represented with dashed arrows, message connecting objects represent the exchange of information between two participants;
- **data associations**: represented with dotted arrows, data association connecting objects put data elements — such as data stores and data objects — in relation to activities.

In other words, sequences establish dependencies among flow objects, while data stores, data objects, and messages define the flow of information. Messages and data objects are transient (i.e., they exist only during the execution of the workflow), while data stores are persistent (i.e., they already exist before the execution of the workflow and continue to exist after the execution of the workflow). Flow and connecting objects are often grouped in a *lane* representing an active agent (either a user or software) in charge of carrying out a portion of the workflow. More lanes (e.g., different agents) may be logically grouped under a *pool* (e.g., an organization). Finally, workflows are usually encoded in XML documents. For more details on BPMN, we refer the interested reader to [47].

**The Pizza Collaboration Workflow.** We report in Figure 3.1 an example workflow from an official OMG report [46]. The workflow models the interaction between a pizza vendor and a pizza customer. The entry point is the "Hungry for Pizza" event, which leads the pizza customer to select and order a pizza with the "Select a pizza" and "Order a pizza" activities, respectively. Afterward, the pizza customer reaches an event-based gateway; this gateway indicates that the pizza customer waits for one of two events that could happen next: either the pizza is received or there is no delivery within 60 minutes, i.e., after one hour the customer calls the pizza vendor and asks for the pizza. In response to this call, the clerk promises that the pizza will be delivered soon, and the customer goes back to waiting. After the "Order a pizza" activity, a message "pizza order" is sent toward the start event "Order Received" activating the "Clerk" lane. The parallel gateway splits the process into concurrent activities, one answering the pizza customer's call and the other one reaching the "Pizza Chef" lane. After the chef baked the pizza, the delivery boy delivers the pizza and receives the payment, which includes giving a receipt to the customer, and reaches an intermediate end event (bold circle with a black dot). Finally, the customer can eat her pizza, reaching the end event (circle with a thick line).

### 3.1.2 Petri and Workflow Nets

Petri nets are widely used to model the flow of objects or information in (concurrent) applications [28]. A Petri net is a tuple $\langle P, T, F \rangle$ representing a (possibly cyclic) directed graph whose vertices correspond to a set of places $P$ (represented as circles) and a set of transitions $T$ (represented as rectangles), while edges correspond to a set of arcs $F \subseteq (P \cup T) \times (P \cup T)$. Petri nets are bipartite graphs, meaning that edges connect places to transitions and vice-versa only, i.e., $F \cap (P \times P) =$

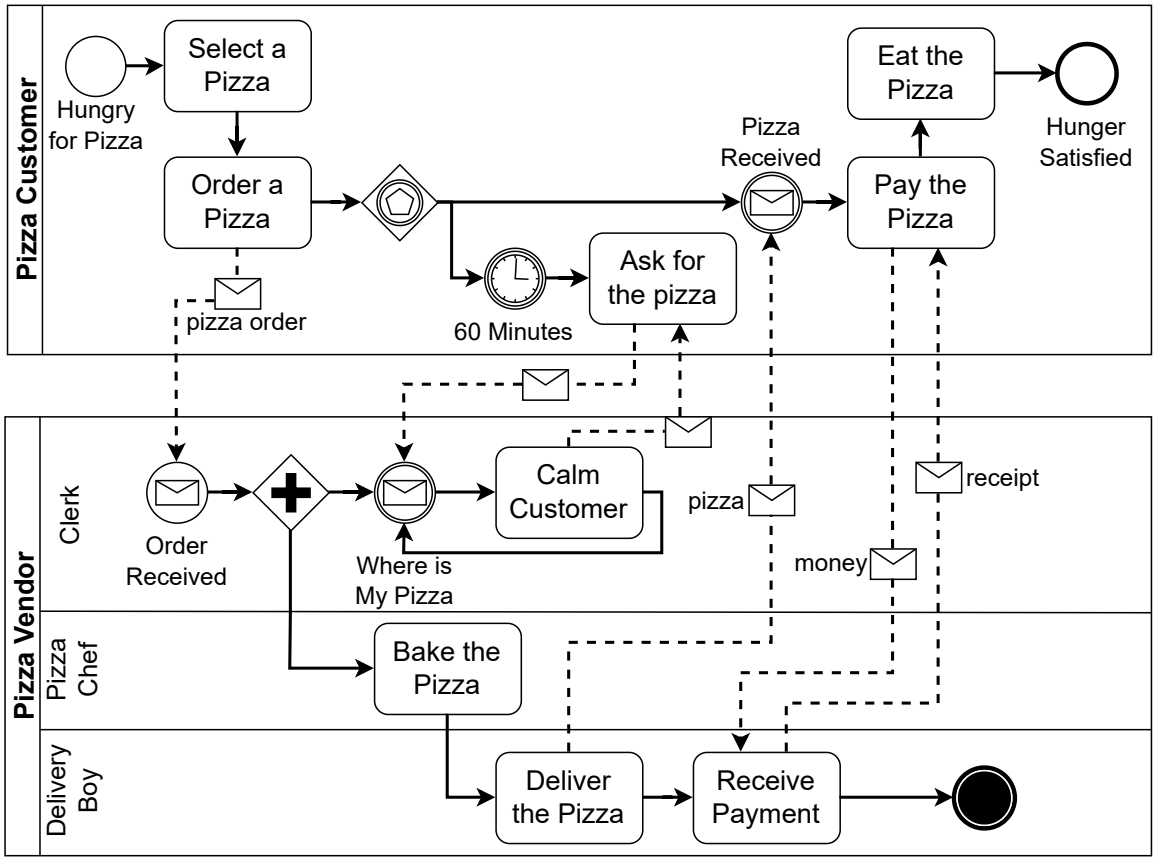


Figure 3.1: BPMN workflow - The Pizza Collaboration

$F \cap (T \times T) = \emptyset$. In other words, a transition is preceded by one or more input places and followed by one or more output places; formally, a transition $t \in T$ has a pre-set ${}^\bullet t = \{p \mid (p,t) \in F\}$ and a post-set $t^\bullet = \{p \mid (t,p) \in F\}$. A place can contain one or more tokens (represented as black full circles) modeling the flow of objects or information. A marking $m$ is a mapping from $P$ to the set of natural numbers $\mathbb{N}$, i.e., defining how many tokens each place of the Petri net contains. A transition $t \in T$ can be enabled (i.e., visited) only if there is at least one token in each of its input places, i.e., $m(p) > 0 \; \forall p \in {}^\bullet t$ (e.g., *t1* in Figure 3.2a). When a transition $t \in T$ is enabled in a marking $m$ — denoted with $m[t\rangle$ — we remove one token from each $p \in^\bullet t$ and add one token to each $p \in t^\bullet$ (see Figures 3.2a and 3.2b) deriving a marking $m'$ such that:

$$m'(p) = \begin{cases} m(p) - 1 & \text{if } p \in {}^\bullet t \\ m(p) + 1 & \text{if } p \in t^\bullet \\ m(p) & \text{otherwise} \end{cases} \tag{3.1}$$

We denote with $[m\rangle$ the set of all markings that can be recursively (i.e., not only immediately) derived starting from $m$.

WF nets (see Figure 3.2) are a subclass of Petri nets typically used to model control — as well as data [121] — flow in (e.g., BPMN) business processes modeled as workflows. Differently from (more generic) Petri nets, WF nets always have a unique source place $p_{source}$ — i.e., that is not the target of any edge — and a unique sink place $p_{sink}$ — i.e., that is not the source of any edge. Moreover, in a WF net, every place and transition is on (at least one) directed path from the source to the sink place. The marking $m_0$ of a WF net $\langle P,T,F\rangle$, which assigns one token to the source place and zero tokens to all other places — i.e., $m_0(p_{source}) = 1 \wedge m_0(p) = 0 \; \forall p \in P \setminus \{p_{source}\}$ — is called

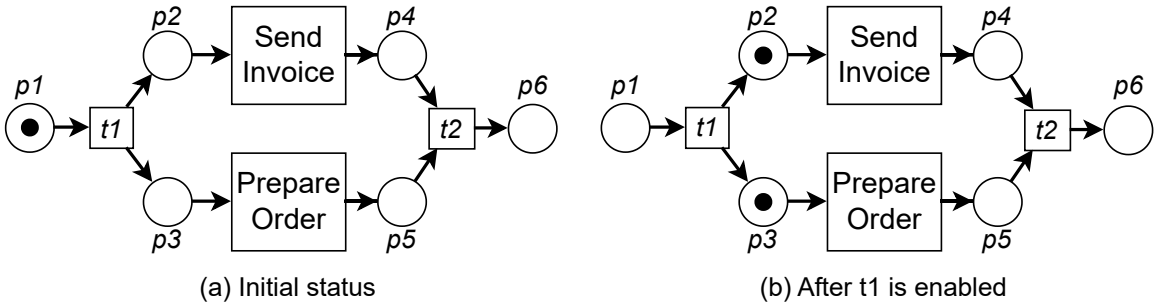


Figure 3.2: Sample WF net

the *canonical marking*. Conversely, the *ending marking* $m_s$ assigns one token to the sink place and zero tokens to all other places — i.e., $m_s(p_{sink}) = 1 \wedge m_s(p) = 0 \; \forall p \in P \setminus \{p_{sink}\}$. WF nets with the canonical marking are *1-safe*, meaning that there cannot be more than one token in a given place at a time [28]. However, WF nets may have multiple transitions that can be enabled simultaneously (e.g., "Send Invoice" and "Prepare Order" in Figure 3.2b); in this case, the choice of which transition to enable first is nondeterministic. Given a WF net $\langle P,T,F \rangle$ with canonical marking $m_0$, we define as a *WF net execution* a sequence of transitions $x = \langle t_1,...,t_n \rangle \mid (t \in T \; \forall t \in x)$ enabled by following the aforementioned marking conditions — that is, $m_0[t_1\rangle \wedge (m'[t_i\rangle \implies m' \in [m\rangle \wedge m[t_{i-1}\rangle \; \forall t_i \in x \setminus \{t_1\})$.

**Soundness.** A WF net $\langle P,T,F \rangle$ with canonical marking $m_0$ satisfies the *soundness* criterion if it has the following two properties [120, 116]:

- **proper termination**: the workflow represented by the WF net can terminate for any case. In other words, for every marking $m$ derived from the canonical marking $m_0$, there exists a transition enabling sequence leading from marking $m$ to marking $m_s$; formally, $m_s \in [m\rangle \; \forall m \in [m_0\rangle$. Moreover, $m_s$ is the only marking that can be derived from $m_0$ that assigns one token to the sink place. Formally, $m \in [m_0\rangle \wedge m(p_{sink}) = 1 \implies m = m_s$;
- **quasi-liveness**: the WF net does not have any *dead transition*. In other words, any transition can possibly be enabled; formally, $\exists m \in [m_0\rangle \mid m[t\rangle \; \forall t \in T$.

A WF net having the *proper termination* property — but not the quasi-liveness property — is said to satisfy the *weak soundness* criterion [116]. Intuitively, the soundness criterion implies the weak soundness criterion. A WF net satisfying weak soundness is guaranteed to always have at least one *complete WF net execution* $x = \langle t_1,...,t_n \rangle$, that is, a WF net execution where exactly one token reaches the sink place; formally, $m[t_n\rangle \implies [m\rangle = \{m_s\}$. A complete WF net execution reflects the execution of the workflow modeled by the WF net; we discuss WF net executions more in detail in Section 3.3.2.

### 3.1.3 Access Control as State Transition Systems

As in [69], we define an *AC scheme* as a state transition system $\langle \Gamma, Q, \vdash, \mathbf{\Psi} \rangle$ where $\Gamma$ is a set of states, $Q$ is a set of queries, $\mathbf{\Psi}$ is a set of state-change rules and $\vdash: \Gamma \times Q \to \{true, false\}$ is the entailment relation determining whether a query is true in a given state. We can instantiate an AC scheme into an *AC system* which consists of a pair $\langle \gamma, \Psi \rangle$ where $\gamma \in \Gamma$ and $\Psi \subseteq \mathbf{\Psi}$. Given an AC system $\langle \gamma, \Psi \rangle$, a sequence of state-change rules $\langle \psi_1,...,\psi_{n-1} \rangle \mid \psi_i \in \Psi$ for $0 < i < n$ induces an *AC execution* $\langle \gamma_1, \psi_1, \gamma_2, ..., \gamma_{n-1}, \psi_{n-1}, \gamma_n \rangle$ where $\gamma_1 = \gamma$ and $\gamma_i \mapsto_{\psi_i} \gamma_{i+1}$ — i.e., $\psi_i$ transforms $\gamma_i$ into $\gamma_{i+1}$ — for $0 < i < n$.

**Role-based Access Control.** In the rest of the chapter, we consider policies specified in RBAC because of its wide adoption in both academia and industry [19] and the fact that business workflows are usually complemented with RBAC policies (see Section 3.1.1). In RBAC, *users* are assigned to one or more *roles* — in the context of an organization, a *role* usually 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, in RBAC a state $\gamma \in \Gamma$ is described as a tuple $\langle \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{UR}, \mathbf{PA} \rangle$, where $\mathbf{U}$ is the set of users, $\mathbf{R}$ is the set of roles, $\mathbf{F}$ is the set of resources, $\mathbf{UR} \subseteq \mathbf{U} \times \mathbf{R}$ is the set of user-role assignments and $\mathbf{PA} \subseteq \mathbf{R} \times \mathbf{PR}$ is the set of role-permission assignments, being $\mathbf{PR} \subseteq \mathbf{F} \times \mathbf{OP}$ a derivative set of $\mathbf{F}$ combined with a fixed set of actions $\mathbf{OP}$ (e.g., read, write). Both $\mathbf{OP}$ and $\mathbf{PR}$ are not part of the state, as $\mathbf{OP}$ remains constant over time and $\mathbf{PR}$ is derivative of $\mathbf{F}$ and $\mathbf{OP}$. We note that role hierarchies can always be compiled away by adding suitable pairs to $\mathbf{UR}$. The set of all queries is $Q = \mathbf{U} \times \mathbf{PR}$, thus a query $q = \langle u, \langle f, op \rangle \rangle$ means whether the user $u \in \mathbf{U}$ can use the permission $\langle f, op \rangle \in \mathbf{PR}$, where $f \in \mathbf{F}$ and $op \in \mathbf{OP}$. We note that, when considering $\mathbf{OP} = \{\texttt{read}, \texttt{write}\}$, there exist two types of queries, i.e., to read and write over a resource, respectively. We report the complete set $\mathbf{\Psi}$ of state-change rules for RBAC — according to the NIST [37] — and the resulting states in Table 3.2. For the sake of simplicity, we consider the entailment relation $\gamma \vdash (\langle u, \langle f, op \rangle \rangle)$ as a special state-change rule for RBAC that does not transform the state $\gamma$ to which it is applied — i.e., $\gamma \mapsto_{\gamma \vdash (\langle u, \langle f, op \rangle \rangle)} \gamma$ — but has the side effect of evaluating the query $\langle u, \langle f, op \rangle \rangle$. In other words, the entailment corresponds to users' AC requests, while the other state-change rules for RBAC correspond to administrative AC requests. Hereafter, when mentioning state-change rules, we refer to state-change rules for RBAC only.

Table 3.2: The set $\mathbf{\Psi}$ of state-change rules for a generic RBAC scheme [37]

| State-Change Rule $\psi_i$ | Resulting AC Policy State $\gamma_{i+1}$ |
|---|---|
| addUser($u$) | $\langle \mathbf{U} \cup \{u\}, \mathbf{R}, \mathbf{F}, \mathbf{UR}, \mathbf{PA} \rangle$ |
| deleteUser($u$) | $\langle \mathbf{U} \setminus \{u\}, \mathbf{R}, \mathbf{F}, \mathbf{UR}, \mathbf{PA} \rangle$ |
| addRole($r$) | $\langle \mathbf{U}, \mathbf{R} \cup \{r\}, \mathbf{F}, \mathbf{UR}, \mathbf{PA} \rangle$ |
| deleteRole($r$) | $\langle \mathbf{U}, \mathbf{R} \setminus \{r\}, \mathbf{F}, \mathbf{UR}, \mathbf{PA} \rangle$ |
| addResource($f$) | $\langle \mathbf{U}, \mathbf{R}, \mathbf{F} \cup \{f\}, \mathbf{UR}, \mathbf{PA} \rangle$ |
| deleteResource($f$) | $\langle \mathbf{U}, \mathbf{R}, \mathbf{F} \setminus \{f\}, \mathbf{UR}, \mathbf{PA} \rangle$ |
| assignUserToRole($u, r$) | $\langle \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{UR} \cup \{(u, r)\}, \mathbf{PA} \rangle$ |
| revokeUserFromRole($u, r$) | $\langle \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{UR} \setminus \{(u, r)\}, \mathbf{PA} \rangle$ |
| assignPermissionToRole($r, \langle f, op \rangle$) | $\langle \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{UR}, \mathbf{PA} \cup \{(r, \langle f, op \rangle)\} \rangle$ |
| revokePermissionFromRole($r, \langle f, op \rangle$) | $\langle \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{UR}, \mathbf{PA} \setminus \{(r, \langle f, op \rangle)\} \rangle$ |
| $\gamma \vdash (\langle u, \langle f, op \rangle \rangle) : true \iff \exists r \in \mathbf{R} \mid (u, r) \in \mathbf{UR} \wedge (r, \langle f, op \rangle) \in \mathbf{PA}$ | |

Table 3.3: RBAC policy state blueprints from [35]

| State | Set | | | | | $r \to u$ | | $u \to r$ | | $\langle f, op \rangle \to r$ | | $r \to \langle f, op \rangle$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Blueprint | $\lvert U \rvert$ | $\lvert R \rvert$ | $\lvert P \rvert$ | $\lvert UR \rvert$ | $\lvert PA \rvert$ | $M$ | $m$ | $M$ | $m$ | $M$ | $m$ | $M$ | $m$ |
| domino | 79 | 20 | 231 | 75 | 629 | 3 | 0 | 30 | 1 | 209 | 1 | 10 | 1 |
| emea | 35 | 34 | 3,046 | 35 | 7,211 | 1 | 1 | 2 | 1 | 554 | 9 | 31 | 1 |
| firewall1 | 365 | 60 | 709 | 1,130 | 3,455 | 14 | 0 | 174 | 1 | 617 | 1 | 25 | 1 |
| firewall2 | 325 | 10 | 590 | 325 | 1,136 | 1 | 1 | 222 | 1 | 590 | 6 | 8 | 1 |
| healthcare | 46 | 13 | 46 | 55 | 359 | 5 | 1 | 17 | 1 | 45 | 7 | 12 | 1 |
| university | 493 | 16 | 56 | 495 | 202 | 2 | 1 | 288 | 1 | 40 | 2 | 12 | 1 |

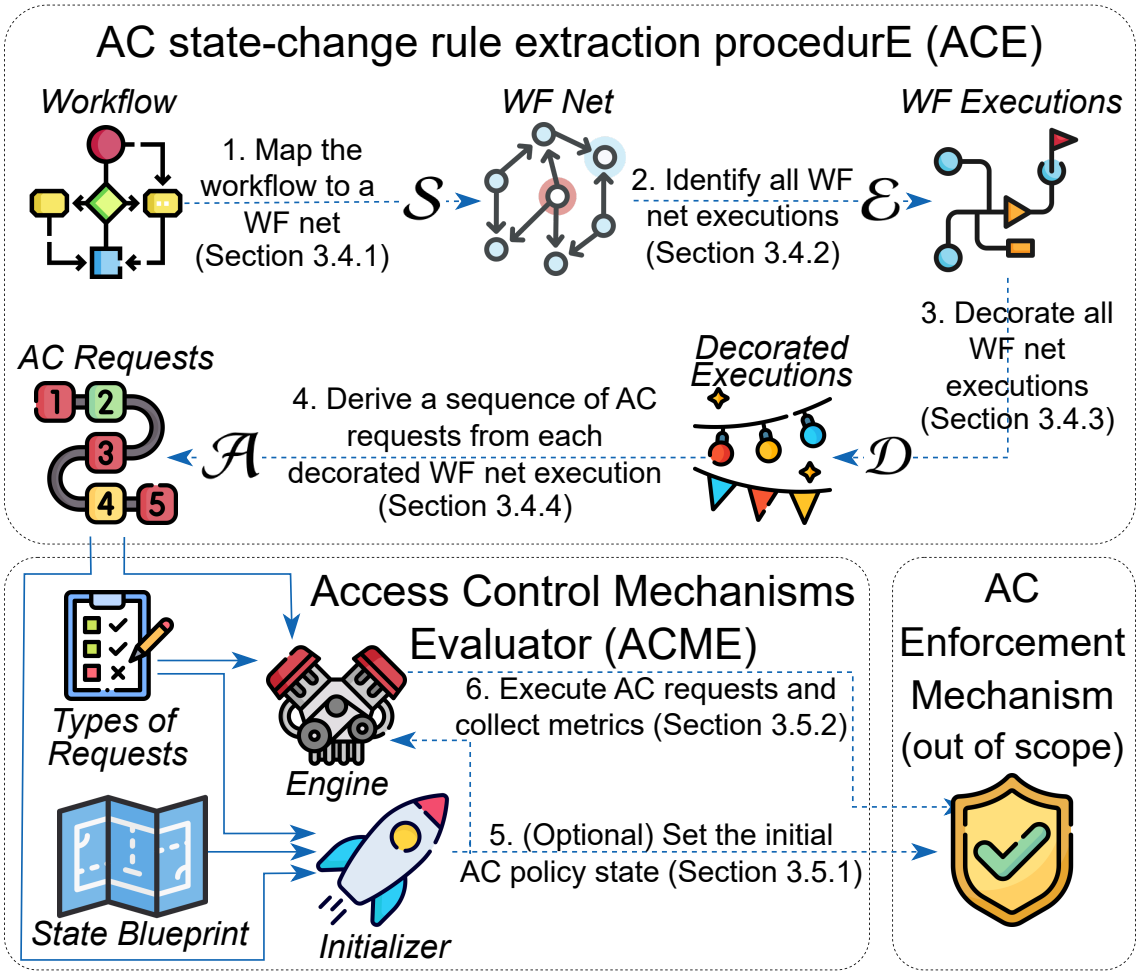


Figure 3.3: High-level graphical representation of our methodology

**Role-based Access Control Policy State Blueprints.** A *state blueprint* for RBAC consists of a tuple $\langle |\mathbf{U}|, |\mathbf{R}|, |\mathbf{F}|, |\mathbf{UR}|, |\mathbf{PA}| \rangle$ — expressing the cardinality of each set of an RBAC state — along with a set of integers representing the maximum ($M$) and minimum ($m$) number — but not the identifiers — of roles assigned to each user ($r \to u$), users assigned to each role ($u \to r$), permissions assigned to each role ($\langle f, op \rangle \to r$) and roles assigned to each permission ($r \to \langle f, op \rangle$). Ene et al. [35] present some RBAC state blueprints mined from real-world datasets which we report in Table 3.3. For instance, the *healthcare* state blueprint expects an RBAC policy with 46 users, 13 roles, and 46 resources, where (in total) there are 55 assignments between users and roles and 359 assignments between roles and permissions. However, each user can be assigned to at least 1 role and at most 5 roles (i.e., $r \to u$), while each role can be assigned to at least 1 user and at most 17 users (i.e., $u \to r$). Finally, each role can be assigned to at least 7 permissions and at most 45 permissions (i.e., $\langle f, op \rangle \to r$), while each permission can be assigned to at least 1 role and at most 12 roles (i.e., $r \to \langle f, op \rangle$). We show how a state blueprint can be used to generate a state in $\Gamma$ in Section 3.4.1.

## 3.2 Methodology Overview

We now give an overview of our methodology, for which we report a graphical representation in Figure 3.3.[4] Our methodology comprises two security mechanisms, namely ACE to derive representative

[4]Icons by SBTS2018, mj, IconHome, Vignesh Oviyan, Freepik from flaticon.com

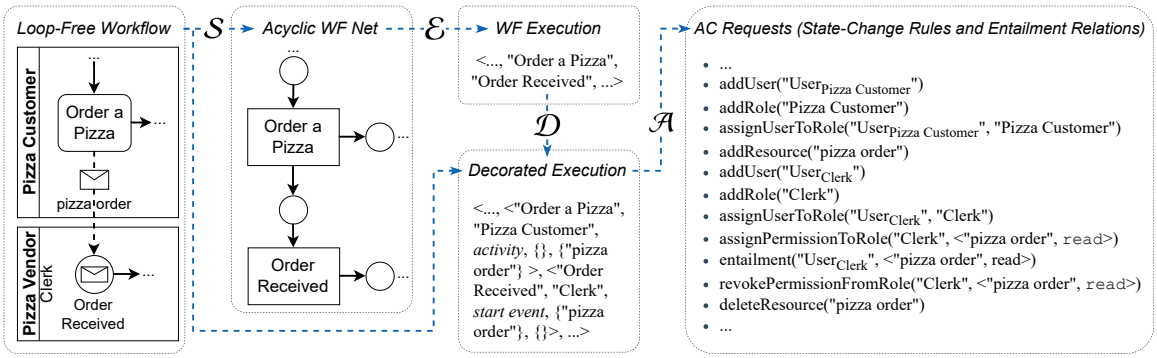


Figure 3.4: Application of ACE on (a portion of) The Pizza Collaboration BPMN workflow

sequences of AC requests from workflows (Section 3.2.1), and ACME, using these sequences to evaluate the performance of AC enforcement mechanisms (Section 3.2.2). Below, we keep the discussion at a high level to allow the reader to get a general understanding of our methodology before delving into (more complex) details; we provide a meticulous description of ACE and ACME in Sections 3.3 and 3.4, respectively. Finally, we discuss the main issues affecting experimental evaluations using micro-benchmarks to (try to) measure the performance of AC enforcement mechanisms realistically and argue how our methodology solves these issues (Section 3.2.3). Throughout this section, we formulate 5 assumptions (*A*) which we group and report in Table 3.4 for ease of reference; we discuss each assumption as soon as we introduce it in the text below. While *A1*, *A2*, and *A5* are key for our methodology, it is possible to circumvent *A3* and *A4* if needed, as we later explain in Section 3.3.4.

### 3.2.1 ACE Overview

As shown in Figure 3.3, ACE works in 4 steps: map to WF nets, identify WF net executions, decorate WF net executions and derive AC requests. Below, we briefly present each step, using the workflow in Figure 3.1 as a running example by reporting in Figure 3.4 the corresponding transformations.

**Map to Workflow Nets - $\mathcal{S}$.** Given as input a BPMN-compliant workflow describing a business process (*A1*), we derive a WF net (step 1 in Figure 3.3) by relying on a well-established stream of work proposing a formal semantics — as pioneered by [28] — whereby flow objects (e.g., activities, gateways) and connecting objects (e.g., sequences, messages, and data associations) of the workflow are mapped to vertices (i.e., transitions and places) and edges of the WF net (see $\mathcal{S}$ in Figure 3.4). We assume the input workflow to not contain any loop (*A2*) as, intuitively, the sequences of AC requests we derive need to be finite to be used by ACME; noncyclicality of workflows can be achieved by design or, e.g., by pre-processing workflows to unroll loops by repeating them a specific number of times. The output of this step is therefore an acyclic WF net $\langle P, T, F \rangle$. We discuss the mapping more in detail in Section 3.3.1.

Table 3.4: Assumptions on workflows and AC enforcement mechanisms

| *A* | Description |
|---|---|
| *A1* | Workflows are written in standard BPMN |
| *A2* | Workflows do not contain loops |
| *A3* | Workflows contain messages and data associations objects |
| *A4* | Workflows group activities under pools and lanes |
| *A5* | Mechanisms implement the state-change rules in Table 3.2 |

**Identify Workflow Net Executions - $\mathcal{E}$.** Given as input an acyclic WF net, we compute its topological ordering to identify all WF net executions (see the definition in Section 3.1.2) which express all possible ways to achieve the business goal modeled in the corresponding workflow (step 2). A topological order of a generic directed acyclic graph — like a WF net — is a linear ordering of its vertices such that for every edge $\langle v, v' \rangle$ from vertex $v$ to vertex $v'$ it holds that $v$ comes before $v'$ in the ordering. In other words, a topological order is a traversal of the graph in which each vertex is visited only after all its dependencies are visited. We highlight that, while computing the topological ordering of a WF net, we still have to respect the requirements for enabling transitions (see Section 3.1.2). Intuitively, this may lead to WF net executions that do not contain all transitions, as it is in the nature of WF nets that not all transitions are enabled; this may happen when, e.g., the original workflow contains exclusive gateways (see Section 3.1.1). Nonetheless, we are guaranteed that every transition is part of (at least one) WF net execution (recall the definition of WF net in Section 3.1.2). The output of this step is the list of (unique) WF net executions, where we label each transition with the name of the corresponding BPMN flow object (see $\mathcal{E}$ in Figure 3.4). We discuss the identification of WF net executions more in detail in Section 3.3.2.

**Decorate Workflow Net Executions - $\mathcal{D}$.** We decorate each WF net execution previously found with further information extracted from the original workflow (step 3); we later use the information to derive AC requests. In particular, we associate to each transition — derived from a BPMN flow or connecting object (see [28]) — the pool or lane that contains the object (*A4*) and the type of the object itself. For flow objects, we also collect the identifiers of incoming and outgoing data (e.g., associations and messages) into two sets (*A3*). The output of this step is the list of decorated WF net executions (see $\mathcal{D}$ in Figure 3.4). We discuss the decoration of WF net executions more in detail in Section 3.3.3.

**Derive Access Control Requests - $\mathcal{A}$.** We translate each decorated WF net execution given as input into a sequence of AC requests (step 4), where each AC request is a state-change rule (see Table 3.2) — in other words, AC requests and state-change rules map to the same concept. We highlight that the state-change rules are not manually inferred from the semantics of the transitions (e.g., the description of the activities of the original workflow), but instead they are automatically derived from the decorated WF net execution. More in detail, given a decorated WF net execution, we consider WF net transitions associated with incoming and outgoing data; the presence of such transitions corresponds to either the creation (e.g., for transient messages and data objects), modification (e.g., for persistent data stores) or reading of resources, for which a permission may be required and checked by an AC enforcement mechanism. Since business workflows are usually complemented with RBAC policies (see Section 3.6.1), we distribute and revoke permissions through roles, which we identify with BPMN pools and lanes. The output of this step is, for each decorated WF net execution, a sequence of state-change rules (see $\mathcal{A}$ in Figure 3.4). We discuss the derivation of state-change rules from decorated WF net executions more in detail in Section 3.3.4.

### 3.2.2 `ACME` Overview

We assume an AC enforcement mechanism supporting RBAC to be capable of processing (at least) the state-change rules reported in Table 3.2; in other words, a mechanism should allow manipulating the state of the RBAC policy (i.e., administrative AC requests) and evaluating the entailment relation (i.e., users' AC requests) — at least for read and write actions over a resource (*A5*). Under this assumption, we note that `ACME` can interface with different kinds of mechanisms and it is not limited to OPA, XACML, and `CryptoAC`. `ACME` also allows organizations to specify which state-change rules

to consider during the performance evaluation (i.e., the *Types of Requests* in Figure 3.3); for instance, an organization may be interested in evaluating a mechanism based on users' but not administrative AC requests. As shown in Figure 3.3, ACME is composed of two modules, initializer and engine, which we describe below.

**Initializer.** This module allows deriving an initial RBAC policy state for the mechanism under evaluation (step 5). The initializer takes three inputs: (*i*) the sequences of state-change rules obtained from ACE, (*ii*) the set of state-change rules $\Psi \subseteq \mathbf{\Psi}$ to consider (i.e., the *Types of Requests* in Figure 3.3), and (*iii*) an RBAC state blueprint (see Section 3.1.3). Alternatively from what is represented in Figure 3.3, instead of using the initializer, organizations can autonomously define an RBAC state representative of their internal structure — e.g., number of employees and qualifications.

**Engine.** This module measures the performance of the mechanism under evaluation by simulating the execution of workflows to obtain representative results (step 6). The engine takes three inputs: (*i*) an RBAC policy state $\gamma \in \Gamma$ determined by either the initializer or the organization, (*ii*) the set of state-change rules $\Psi \subseteq \mathbf{\Psi}$ to consider (i.e., the *Types of Requests* in Figure 3.3), (*iii*) and the sequences of state-change rules produced by ACE (see Section 3.2.1). We highlight how the first two inputs (i.e., $\langle \gamma, \Psi \rangle$) constitute an RBAC system (recall the definition from Section 3.1.3). This RBAC system is used by the engine to induce AC executions from the sequences of state-change rules produced by ACE, reflecting the execution of the corresponding workflows. Finally, the engine runs (possibly in a concurrent and intertwined fashion) the obtained AC executions by invoking the functionalities of the mechanism accordingly. As shown in Figure 3.3, the configuration and deployment of the mechanism are out of scope.

### 3.2.3 Our Methodology and Micro-benchmarking

Micro-benchmarks are traditionally implemented by sending single (either administrative or users') AC requests to the mechanism being evaluated sequentially, as said in Section 3.6.2, while possibly scaling specific aspects of the AC policy state, like the number of roles (e.g., [9]) or attributes (e.g., [102, 76]). Although micro-benchmarks are surely useful to evaluate single functionalities of a mechanism, we believe that two fundamental requirements to obtain truly representative results for a specific scenario are to (*i*) consider a realistic AC policy state and (*ii*) expect multiple users to send AC requests concurrently; our methodology satisfies both. Potentially, also micro-benchmarks can satisfy these requirements; indeed, it is ideally possible to run multiple instances of a micro-benchmark concurrently and fix the AC policy to a realistic state. However, to the best of our knowledge, there are no such examples in the literature.

Nonetheless, we identify two structural issues in micro-benchmarks. First, micro-benchmarks are likely to activate optimizations, even though indirectly, that may alter the significance of the final results. The clearest example is caching, which allows a mechanism to answer more quickly to an (unrealistic) sequence of identical AC requests of the same type and parameters, as argued for XACML in [56, 16]. Caching applies both at the application level and to, e.g., CPU and RAM. Furthermore, given that some mechanisms may benefit from this kind of (unrealistic) caching more than others, we conclude that results obtained with micro-benchmarks can hardly be representative of a realistic scenario. By considering different kinds of AC requests (i.e., those discussed in Section 3.2.2), our methodology is way less likely to activate such optimizations and, in any case, those optimizations would be the same as — or at least similar to — those activated in a realistic scenario. The second issue concerns the typical usage of an AC enforcement mechanism, which largely depends on the business processes (i.e., the workflows) carried out by an organization. For

instance, consider an organization that has to decide which AC enforcement mechanism to deploy between two mechanisms, $M_1$ and $M_2$. Now, assume that a micro-benchmark evaluation reports generally better results for $M_1$ over $M_2$, except for a few types of AC requests. Based on these results, the organization may then choose $M_1$ over $M_2$. However, this choice may be wrong, as the real answer depends on how frequently the types of AC requests in which $M_2$ performs better are present in the business processes — thus, in the workflows — of the organization. Therefore, contrary to what intuitively results from the micro-benchmark evaluation, $M_2$ may be a better fit for the specific scenario of the organization than $M_1$, even though generally less performing; our methodology is designed to address exactly this issue. We provide a concrete example of this situation in Section 3.5.4.

## 3.3 ACE

We now present our procedure, ACE, to extract sequences of state-change rules from workflows. First, we explain how to obtain a WF net from a workflow (Section 3.3.1). Then, we show how to identify all WF net executions (Section 3.3.2) and describe how to decorate them with information (Section 3.3.3) that we later use to extract sequences of state-change rules which are representative of the original workflow (Section 3.3.4). We represent each transformation as a function in Table 3.5; their nested invocation — which corresponds to ACE — precisely defines how to derive realistic sequences of state-change rules (i.e., those in Table 3.2) from a workflow specified in BPMN. Finally, we present the implementation of ACE (Section 3.3.5).

### 3.3.1 Map to Workflow Nets - $\mathcal{S}$

Dijkman et al. [28] use WF nets to provide a formal semantics to (the execution of) BPMN workflows. This semantics defines a *WF module* for several common BPMN flow objects (e.g., activities, events, and gateways) and connecting objects (e.g., sequence and messages). For instance, the semantics maps an activity onto a module composed of a transition with one input place and one output place; intuitively, the transition models the completion of the activity. Similarly, the semantics maps a message onto a module composed of a transition with one input place and one output place; intuitively, the transition models the transmission of the message. For further examples and more details on the semantics, we refer the interested reader to [28]. In our work, we use the proposed semantics to map BPMN flow and connecting objects to transitions and places. We denote with $\mathcal{S}$ (see Table 3.5) the function that takes as input a BPMN workflow $W$ — where loops were either absent or previously unrolled, as discussed in Section 3.2.1 — and returns an acyclic WF net $(P,T,F)$ using — hence, inheriting the accuracy and completeness of — the formal semantics in [28]; in other

Table 3.5: Composition of ACE

| Function | Input | Output | Description |
|---|---|---|---|
| $\mathcal{S}$ | $W$ | $\langle P,T,F\rangle$ | Obtain an acyclic WF net $\langle P,T,F\rangle$ from the (loop-free) BPMN workflow $W$ using the formal semantics in [28] |
| $\mathcal{E}$ | $\langle P,T,F\rangle$ | $X$ | Identify the set of all (complete) WF net executions $X$ of the WF net $\langle P,T,F\rangle$ by repeatedly invoking Algorithm 4 |
| $\mathcal{D}$ | $W,X$ | $X^d$ | Derive the set of decorated WF net executions $X^d$ from the WF net executions $X$ of $W$ by repeatedly invoking Algorithm 5 |
| $\mathcal{A}$ | $X^d$ | $C$ | Derive the list of sequences of state-change rules $C$ from the decorated WF net executions $X^d$ by repeatedly invoking Algorithm 6 |
| | | | $\texttt{ACE}(W) \coloneqq \mathcal{A}(\mathcal{D}(W,\mathcal{E}(\mathcal{S}(W))))$ |

words, (the execution of) the WF net $(P,T,F)$ is representative of (the execution of) the business process modeled by $W$.

### 3.3.2 Identify Workflow Net Executions - $\mathcal{E}$

Given an acyclic WF net $(P,T,F)$, we compute its topological ordering to identify all WF net executions. In detail, we modify an algorithm for topological ordering based on depth-first search [26] according to the peculiarities of WF nets (see Section 3.1.2) and report it in Algorithm 4. The algorithm takes as input an acyclic WF net $(P,T,F)$ and returns a topological order $o$, which consists of an ordered sequence of $k$ transitions and places, i.e., $o = \langle v_1,...,v_k \rangle \mid v \in (P \cup T)\ \forall v \in o \wedge v_1 = p_{source}$. As it is possible to see from Algorithm 4, our modification to the original approach in [26] is straightforward, and consists in starting the visit from $p_{source}$ by considering the canonical marking $m_0$ (line 3 in Algorithm 4) and then visiting a transition — and consequently deriving a new marking — only when the transition is enabled (lines 15—22). As mentioned in Section 3.2.1, a topological order for a WF net may not contain all vertices of the WF net, as not every transition is always enabled. Nonetheless, we are certain that Algorithm 4 always terminates as (*i*) the input WF net is acyclic — that is, it contains no loops — and (*ii*) every vertex of the WF net is on (at least one) directed path *from the source to the sink place* (see Section 3.1.2), i.e., visiting a vertex always brings the visit closer to the sink place. A corollary of (*ii*) is that Algorithm 4 exhaustively finds all WF net executions, under the assumption that all possible markings are considered. More concretely, given a topological order $o = \langle v_1,...,v_k \rangle$ of a WF net $(P,T,F)$, we obtain a WF net execution $x = \langle t_1,...,t_n \rangle$ by considering all and only transitions in $o$ (i.e., $t_i \in x \iff (\exists v \in o \mid v \in T \wedge t_i = v)$ for $0 < i < n$) while preserving the same ordering (i.e., $(t_i,t_j \in x \wedge 0 < i < j \leq n) \iff (\exists v_h,v_l \in o \mid t_i = v_h \wedge t_j = v_l \wedge 0 < h < l \leq k)$.

The fact that Algorithm 4 always terminates does not necessarily imply that each WF net execution is complete, i.e., correctly captures a way to achieve the modeled business goal; indeed, we are not guaranteed that the WF net satisfies the soundness criterion (recall the definition of

**Algorithm 4:** WF net topological ordering

**Input:** $(P,T,F)$
**Output:** $o/\perp$

```
1  let A = ∅; // set of marked vertices
2  let o = []; // array for the topological order
3  let m = m_0; // start from the canonical marking
4  while ∃v ∈ (P ∪ T) | v ∉ A ∧ (v ∈ P ∨ (m(p) = 1 ∀p ∈ •v)) do
5  |  invoke visit(v);
6  end
7  if m(p_sink) ≠ 1 then
8  |  return ⊥;
9  end
10 return o;

11 Function visit(v):
12 |  if v ∈ A then
13 |  |  return;
14 |  end
15 |  if v ∈ T then
16 |  |  if m(p) > 0 ∀p ∈ •v // enable the transition
17 |  |  then
18 |  |  |  m(p) = m(p) − 1 ∀(p ∈ •v);
19 |  |  |  m(p) = m(p) + 1 ∀(p ∈ v•);
20 |  |  else
21 |  |  |  return;
22 |  |  end
23 |  end
24 |  invoke visit(u) ∀u ∈ (P ∪ T) | ⟨v,u⟩ ∈ F;
25 |  add v to A and add v to the head of o;
```

these notions in Section 3.1.2). Determining the (even weak) soundness of an acyclic WF net is a co-NP-complete problem [116] that falls within the competence area of (BPMN) workflow designers. Hence, this problem is out of the scope of our work; we refer the interested reader to [116, 121, 120]. Here, we just consider complete WF net executions (lines 7—10). Finally, we denote with $\mathcal{E}$ (see Table 3.5) the function that takes as input a WF net $(P,T,F)$ and returns the set of all (complete) WF net executions $X$ by repeatedly invoking Algorithm 4 considering all possible markings (i.e., $[m_0\rangle$). We note that we can easily set an upper bound to the number of markings we consider, if needed.

### 3.3.3 Decorate Workflow Net Executions - $\mathcal{D}$

After having computed the set of (complete) WF net executions $X$ through the topological ordering of the corresponding WF net $(P,T,F)$, we decorate each WF net execution with further information that we later use to extract sequences of state-change rules. We report the pseudocode for the decoration of a WF net execution in Algorithm 5. Given a WF net execution $x = \langle t_1, ..., t_n \rangle$ of a WF net derived from a workflow $W$ ($W$ is a required input as it contains the decorating information returned by the functions in lines 15—18 in Algorithm 5), we attach to each transition $t \in x$ — where $t$ was derived from a flow or connecting object with symbol $s$ (recall the definitions of these notions in Section 3.1.1) — the pool or lane $l$ that contains $s$ and $s$ itself, obtaining a tuple $\langle t,l,s \rangle$ (lines 3—4). If $s$ is a flow object, we also collect the identifiers of data (associations and messages) incoming to and outgoing from $s$ into two sets: one for incoming (i.e., required) data $req_t$ and one for outgoing (i.e., produced) data $prod_t$, respectively (lines 6—7). Summarizing, we attach $l$, $s$, $req_t$ and $prod_t$ to $t$; by abusing notation, we write $\langle t,l,s,req_t,prod_t \rangle$. A decorated WF net execution $x^d$ obtained from a WF net execution $x$ is thus of the form $\langle \langle t_1,l_1,s_1,req_{t_1},prod_{t_1} \rangle, ..., \langle t_n,l_n,s_n,req_{t_n},prod_{t_n} \rangle \rangle$. We denote with $\mathcal{D}$ (see Table 3.5) the function that takes as input a BPMN workflow $W$ and the set of WF net executions $X$ of the corresponding WF net, and returns the set of decorated WF net executions $X^d$ by repeatedly invoking Algorithm 5 on each $x \in X$.

**Algorithm 5:** Decoration of a WF net execution

```
   Input: W, x
   Output: x^d
1  let x^d = []; // array for the decorated transitions
2  foreach t ∈ x do
3  |  let s = findObject(t, W);
4  |  let l = findPoolOrLane(s, W);
5  |  if s is a flow object then
6  |  |  let req_t = findRequiredData(s, W);
7  |  |  let prod_t = findProducedData(s, W);
8  |  else
9  |  |  let req_t = ∅;
10 |  |  let prod_t = ∅;
11 |  end
12 |  add ⟨t, l, s, req_t, prod_t⟩ to x^d;
13 end
14 return x^d;

15 Function findObject(t, W):
   |  // return t's BPMN object in W
16 Function findPoolOrLane(s, W):
   |  // return s's BPMN pool or lane in W
17 Function findRequiredData(s, W):
   |  // return s's set of incoming data in W
18 Function findProducedData(s, W):
   |  // return s's set of outgoing data in W
```

### 3.3.4 Derive Access Control Requests - $\mathcal{A}$

We now discuss how to derive sequences of state-change rules in $\mathbf{\Psi}$ (i.e., those reported in Table 3.2) which are representative of the business processes described by workflows. The ideal scenario would be for workflows to already embed AC information (e.g., describing the permissions of roles on resources). However, even though several researchers proposed to extend BPMN to express security requirements (e.g., [123, 83, 117, 15, 105, 104, 72]), at the time of writing BPMN does not integrate symbols to express AC information. Therefore, as similarly done in the literature (see Section 3.6.1) we rely on the syntax of the modeling language — BPMN, in our case — to extract state-change rules from workflows automatically. We report the pseudocode for the derivation of a sequence of state-change rules $c$ from a decorated WF net execution $x^d$ in Algorithm 6. Similarly to [33, 83], we create a role for each pool or lane, along with a user which we assign to the role (lines 5—9 in Algorithm 6). Once the corresponding pool is concluded — i.e., we reach the end event of the pool — we revoke the user from the role and delete both (lines 10—15). If a workflow does not group activities under pools and lanes (i.e., if *A4* in Table 3.4 is not respected), we can conservatively consider a dedicated user and role for each activity.

We add, read and write over resources according to the identifiers in $req_t$ and $prod_t$, distributing and revoking the corresponding permissions to the involved roles accordingly, as done in [123, 33, 83]

**Algorithm 6:** Derivation of state-change rules

**Input:** $x^d$
**Output:** $c$

```
1  let c = []; // array for state-change rules
2  let r = []; // array for roles already created
3  foreach ⟨t, l, s, req_t, prod_t⟩ ∈ x^d do
4      if s is a flow object then
5          if l ∉ r then
6              add addUser(user_l) to c;
7              add addRole(l) to c;
8              add assignUserToRole(user_l, l) to c;
9              add l to r;
10         else if s = "end event" then
11             add revokeUserFromRole(user_l, l) to c;
12             add deleteRole(l) to c;
13             add deleteUser(user_l) to c;
14             remove l from r;
15         end
16     end
17     foreach f ∈ req_t do
18         let p = ⟨f, read⟩;
19         add assignPermissionToRole(l, p) to c;
20         add entailment ⊢ (⟨user_l, p⟩) to c;
21         add revokePermissionFromRole(l, p) to c;
22         if f is a transient resource then
23             add deleteResource(f) to c;
24         end
25     end
26     foreach f ∈ prod_t do
27         if f is a persistent resource then
28             let p = ⟨f, write⟩;
29             add assignPermissionToRole(l, p) to c;
30             add entailment ⊢ (⟨user_l, p⟩) to c;
31             add revokePermissionFromRole(l, p) to c;
32         else
33             add addResource(f) to c;
34         end
35     end
36 end
37 return c;
```

(lines 17—35). Transient resources (e.g., messages and data objects) are created within the execution of the workflow and deleted once read, while persistent resources (e.g., data stores) are supposed to exist a-priori. Consequently, transient resources can only be added, read, and deleted, while persistent resources can only be read and written over. If a workflow does not contain messages or data associations (i.e., if *A3* in Table 3.4 is not respected), we can conservatively assign a (transient) input and output to each activity, as also done in [33, 119]; each activity would take as input the output of the activities preceding it.

Finally, we denote with $\mathcal{A}$ (see Table 3.5) the function that takes as input the set of decorated WF net executions $X^d$ and returns the list of sequences of state-change rules $C$ by repeatedly invoking Algorithm 6 on each $x^d \in X^d$.

### 3.3.5 Implementation of ACE

We implement ACE (see Table 3.5) in Python — our implementation expects workflows encoded as XML files according to the BPMN standard — as open-source software. Then, we applied ACE on the workflow in Figure 3.1 to derive sequences of state-change rules.[5] As the input workflow needs to be acyclic, we first pre-process it by removing the loops it contains so to respect *A2* in Table 3.4. In detail, we unroll the "Calm Customer"—"Where is My Pizza" loop by repeating it once. Then, we note a circular dependency between the "Pay the Pizza" and "Receive Payment" activities on the "money" and "receipt" messages. We can easily solve this dependency by splitting the "Pay the Pizza" activity into two (sub)activities, the first being the target of the sequence connecting object from the "Pizza received" event and producing the "money" message, and the second being the source of the sequence connecting object toward the "Eat the Pizza" activity and requiring the "receipt" message.

By applying ACE to the (pre-processed) workflow in Figure 3.1 we obtain 50 different WF net executions from which we derive 20 unique sequences of (the same) 54 state-change rules. First, we note that different WF net executions may sometimes lead to the same sequence of state-change rules. This happens when, e.g., WF net executions differ just for the order of transitions which do not yield any state-change rule (e.g., activities that do neither require nor produce any data). Then, as the workflow in Figure 3.1 does not contain any exclusive or inclusive gateway, all 20 sequences have the same 54 state-change rules but differ in their order. Nonetheless, we are still interested in these sequences, as we cannot know a-priori whether the order in which state-change rules are executed affects the performance of an AC enforcement mechanism. For instance, this is often the case in CAC, where the computational effort in distributing secret cryptographic keys depends on the current number of users, roles, and resources; consequently, adding a new role before or after distributing a new cryptographic key has an impact on performance [9].

We highlight that ACE is logically limited to the WF net mapping in [28]. However, it is relatively easy to add WF net modules for further BPMN symbols. Finally, we measure the running time of the implementation of ACE for the workflow in Figure 3.1, finding it to be 0.809s (average of 100 runs) on a device running Ubuntu 20.04 on an Intel(R) Core(TM) i7-11370H and 16GB of RAM. We believe that the running time of ACE is not really relevant, as ACE runs offline (i.e., with no real-time latency requirements) and just once per workflow.

## 3.4 ACME

We now present our evaluator, ACME, for measuring the performance of AC enforcement mechanisms based on a set of lists of sequences of state-change rules $\{C_1, ..., C_n\}$ derived by ACE (see Section 3.3)

[5]The implementation of ACE is available at https://github.com/stfbk/ACE, while the sequences of state-change rules are available at https://github.com/stfbk/ACME/tree/A_Simulation_Framework_Replication_Package

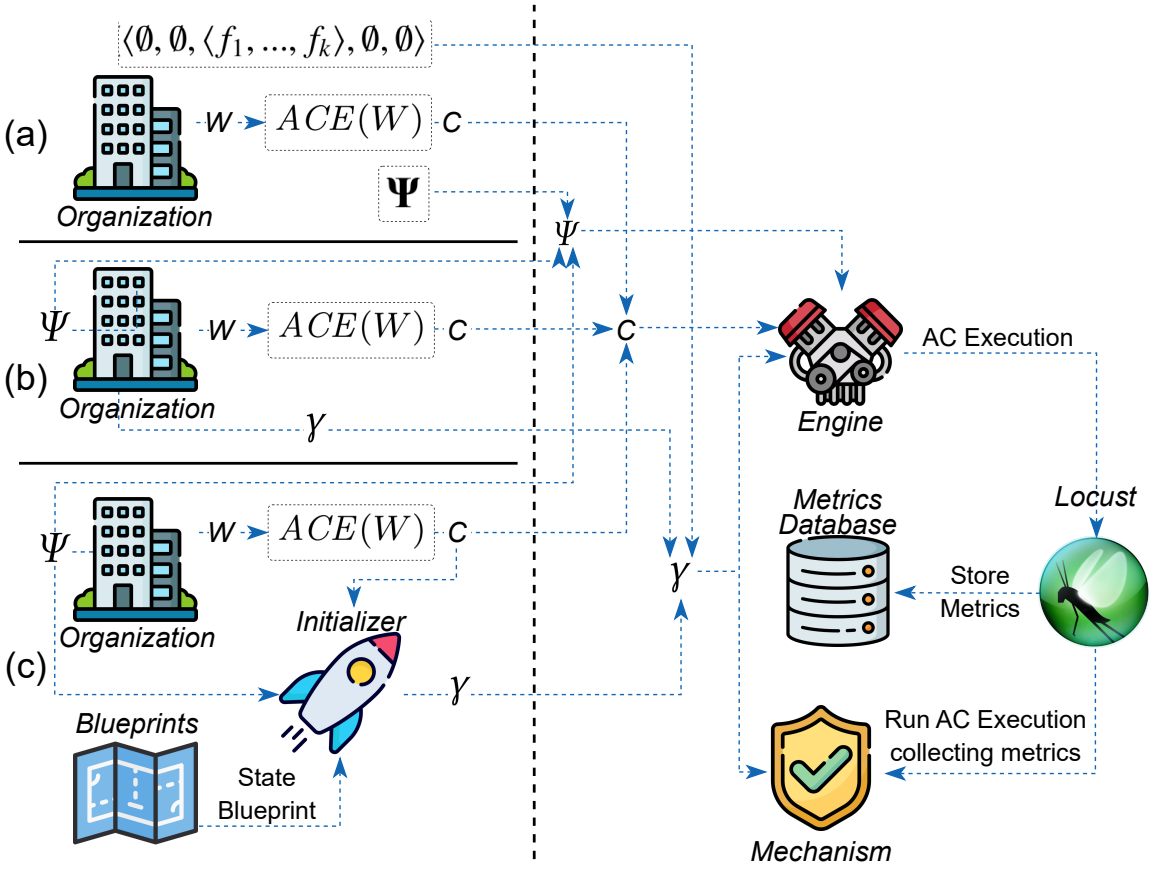


Figure 3.5: Representation of ACME in the context of our methodology for (a) basic, (b) highly customized, and (c) partially customized RBAC System

from a set of BPMN workflows $\{W_1, ..., W_n\}$ — i.e., $C_i = \texttt{ACE}(W_i)$ for $1 \leq i \leq n$. ACME is composed of two modules: the initializer (Section 3.4.1) and the engine (Section 3.4.2). We represent ACME in the context of our methodology — previously shown in Figure 3.3 — in Figure 3.5. Finally, we present the implementation of ACME (Section 3.4.3).

### 3.4.1 Initializing the Mechanism

The inducing of AC executions from $\{C_1, ..., C_n\}$ requires the definition of an RBAC system $\langle \gamma, \Psi \rangle$, as explained in Section 3.1.3. As said in Section 3.2.2, the set of state-change rules $\Psi \subseteq \boldsymbol{\Psi}$ is given by the organization (i.e., the *Types of Requests* in Figure 3.3), while the RBAC policy state $\gamma \in \Gamma$ is determined by either the initializer or the organization. Below, we elaborate on three alternatives for the definition of the RBAC system.

**Basic RBAC System.** A sequence of state-change rules $c \in C$ produced by Algorithm 6 from a workflow $W$ contains — at least potentially — all state-change rules in $\boldsymbol{\Psi}$ (see Table 3.2); in other words, $c$ contains rules for creating (and deleting) users, roles and (transient) resources, and for assigning (and revoking) users and permissions to roles. Indeed, BPMN requires persistent resources only to exist before the execution of the workflow (see Section 3.3.4) — a requirement which is in line with their semantics (see Section 3.1.1). Therefore, the basic (i.e., simplest) RBAC system that can be used to induce an AC execution from $c$ — as shown in Figure 3.5a — is $\langle \langle \emptyset, \emptyset, \langle f_1, ..., f_k \rangle, \emptyset, \emptyset \rangle, \boldsymbol{\Psi} \rangle$, where $k$ is the number of persistent resources in $W$, $f_i$ is a persistent resource of $W$ for $0 < i \leq k$ and $\boldsymbol{\Psi}$ contains all state-change rules in Table 3.2.

**Highly Customized RBAC System.** Evaluating the performance of AC enforcement mechanisms on all state-change rules in $\boldsymbol{\Psi}$ — as it happens when using the basic RBAC system — may be a realistic approach for some scenarios. For instance, a user may be assigned to a role only when needed to carry out a specific task (*run-time role provisioning*) and based on contextual conditions such as environmental attributes (e.g., time of day) and dynamic separation of duty constraints [67]. However, the same approach may be unrealistic for other scenarios. For instance, creating and deleting the "Clerk" role every time a customer orders a pizza in the workflow in Figure 3.1 may not be particularly meaningful or representative of the modeled business process; instead, we could assume that the "Clerk" role already exists in the initial state. Therefore — as shown in Figure 3.5b — an organization can autonomously define the RBAC system $\langle \gamma, \Psi \rangle$ used by ACME to induce AC executions. In other words, an organization can fine-tune the initial state of the RBAC policy — i.e., $\gamma$ — and the subset of state-change rules used to induce AC executions — i.e., $\Psi \subseteq \boldsymbol{\Psi}$. Although providing great flexibility, this definition requires some knowledge and effort on the organization's part, as the organization has to define a state $\gamma$ representative of its internal structure (e.g., number of employees and qualifications) manually or using ad-hoc policy state generation tools — e.g., those presented in [16, 3] specialized for XACML. Moreover, the names (i.e., identifiers) of users, roles, and resources in $\gamma$ need to match those contained in $\{C_1, ..., C_n\}$. Finally, $\gamma$ should be compatible with $\Psi$, i.e., the application of state-change rules in $\Psi$ should result in neither adding elements — i.e., users, roles, resources, assignments, and permissions — already in $\gamma$ nor deleting elements not in $\gamma$; for instance, if $\gamma$ already contains users, then the addUser($\cdot$) state-change rule — where the "$\cdot$" symbol represents a wildcard value — should not be in $\Psi$.

**Partially Customized RBAC System.** As a middle way between a basic RBAC system — i.e., easy definition where an organization specifies neither $\gamma$ nor $\Psi$ — and a highly customized RBAC system — i.e., complex definition where the organization specifies both $\gamma$ and $\Psi$ — ACME provides the organization with an initializer that generates a suitable $\gamma$ automatically. More in detail — as shown in Figure 3.5c — the initializer derives a state $\gamma$ by combining the input set $\Psi$ of state-change rules chosen by the organization and the output of ACE $\{C_1, ..., C_n\}$ with an RBAC state blueprint like those presented in Table 3.3 (see Figure 3.5). In this way, besides the workflows, the organization just sets what state-change rules to use in the performance evaluation (i.e., $\Psi$). For instance, if the organization wants to evaluate the performance of a mechanism on the entailment relation only, then the organization can provide as input $\Psi = \{\vdash (\langle \cdot, \cdot \rangle)\}$, and the initializer can generate a full-fledged $\gamma$ already containing users, roles, resources, assignments, and permissions. Similarly, the set $\Psi = \{$assignPermissionToRole$(\cdot,\cdot)$,revokePermissionFromRole$(\cdot,\cdot)$, assignUserToRole$(\cdot,\cdot)$,revokeUserFromRole$(\cdot,\cdot)\}$ leads to an evaluation based on the assignment (and revocation) of users and permissions to roles only. Using the initializer, the organization can thus decide which state-change rules to consider in AC executions — to a certain extent. Indeed, to prevent inconsistencies, AC executions must have the same initial and final RBAC state; otherwise,

Table 3.6: Partially customized RBAC system constraints

| |
|---|
| $Pr_1(\Psi)$ = addUser$(\cdot)\in\Psi \iff$ deleteUser$(\cdot)\in\Psi$ |
| $Pr_2(\Psi)$ = addRole$(\cdot)\in\Psi \iff$ deleteRole$(\cdot)\in\Psi$ |
| $Pr_3(\Psi)$ = addResource$(\cdot)\in\Psi \iff$ deleteResource$(\cdot)\in\Psi$ |
| $Pr_4(\Psi)$= assignUserToRole$(\cdot)\in\Psi \iff$ revokeUserFromRole$(\cdot)\in\Psi$ |
| $Pr_5(\Psi)$ = assignPermissionToRole$(\cdot)\in\Psi \iff$ revokePermissionFromRole$(\cdot)\in\Psi$ |
| $Pr_6(\Psi)$ = (addUser$(\cdot)\in\Psi \vee$addRole$(\cdot)\in\Psi) \implies$ assignUserToRole$(\cdot)\in\Psi$ |
| $Pr_7(\Psi)$ = (addRole$(\cdot)\in\Psi \vee$addResource$(\cdot)\in\Psi) \implies$ assignPermissionToRole$(\cdot)\in\Psi$ |

it would be impossible to run an AC execution more than once. For instance, if an AC execution creates a role, that role must be deleted before re-running the same AC execution — as otherwise, in the second run, the AC execution would try to create a role that already exists, resulting in an error. Formally, given an RBAC system $\langle \gamma, \Psi \rangle$ and $c = \langle \psi_1, ..., \psi_{n-1} \rangle$ (where $\psi_i \in \Psi$ for $0 < i < n$), $\Psi$ must be defined so to induce an AC execution $\langle \gamma_1, \psi_1, \gamma_2, ..., \gamma_{n-1}, \psi_{n-1}, \gamma_n \rangle$ where $\gamma_1 = \gamma_n = \gamma$. We translate this requirement for $\Psi$ into 7 constraints which we report in Table 3.6 as predicates $Pr_i(\Psi)$ for $1 \leq i \leq 7$. The first 5 predicates state that every element added to the state must also be removed from the state before the end of the AC execution. Then, if users or roles are created dynamically during the execution of a workflow, it naturally follows that assignments of users to roles are created dynamically as well; the same concept applies also to roles and persistent resources (last 2 predicates in Table 3.6). Instead, transient resources can exist within the execution of a workflow only (see Section 3.1.1) and — by definition — can never be part of the initial state $\gamma$. More formally, the organization can pick any one set of state-change rules among those in the set $\{\Psi \mid \Psi \in \mathcal{P}(\mathbf{\Psi}) \wedge Pr_1(\Psi) = Pr_2(\Psi) = Pr_3(\Psi) = Pr_4(\Psi) = Pr_5(\Psi) = Pr_6(\Psi) = Pr_7(\Psi) = true\}$, where $\mathcal{P}(\mathbf{\Psi})$ represents the power set of $\mathbf{\Psi}$.

We report the pseudocode of the initializer in Algorithm 7. First, the initializer checks the consistency of the blueprint, i.e., it asserts that the numbers of users, roles, and resources are compatible with the minimum and maximum numbers of assignments and permissions (line 1 in Algorithm 7), and prepares an empty state (line 2). Then, the initializer populates **U** (lines 3—8) and **R** (lines 9—14) if addUser$(\cdot) \notin \Psi$ and addRole$(\cdot) \notin \Psi$, respectively, with names extracted from $\{C_1, ..., C_n\}$ (lines 43—44). Persistent resources are always present in **F** (lines 15—18 and 45), while transient resources are created during AC executions only. Afterwards, if assignUserToRole$(\cdot, \cdot) \notin \Psi$, the initializer assigns users to roles (i.e., **UR**) according to the input blueprint (lines 19—40). The distribution of permissions of roles over (persistent) resources (i.e., **PA**) follows the same concept — for brevity, we omit it from Algorithm 7.

### 3.4.2 Running the Engine

As shown in Figure 3.5, the engine takes as input an RBAC system $\langle \gamma, \Psi \rangle$ and $\{C_1, ..., C_n\}$ derived from $\{W_1, ..., W_n\}$ as described in Section 3.3. Then, the engine induces an AC execution — containing only state-change rules in $\Psi$ — for each sequence of state-change rules in $\{C_1, ..., C_n\}$, as discussed in Section 3.1.3. Finally, the engine runs each AC execution by invoking — for each state-change rule — the corresponding functionalities (or APIs) of the mechanism being evaluated.

The engine can run one or more AC executions for each $\{C_1, ..., C_n\}$ — i.e., for each workflow — concurrently to simulate the presence of multiple ongoing business processes. The engine also allows defining weights — expressed as natural numbers — to guide the selection of which workflow(s) to instantiate, as to model the fact that some workflows occur more frequently than others. For instance, given $\{C_1, ..., C_n\}$, assigning to $C_i$ a weight $w_{C_i} = 2$ — while $w_{C_j} = 1$ for $1 \leq j \leq n \mid i \neq j$ — means that the workflow $W_i$ corresponding to $C_i$ (i.e., $C_i = \texttt{ACE}(W_i)$) occurs twice as often than the other workflows; organizations can derive values for these weights from, e.g., logs and analytic tools integrating workflow engines. Similarly, the engine can run multiple AC executions of the same workflow concurrently; again, organizations can assign weights to AC executions. In any case, the engine takes care of avoiding inconsistencies in the AC policy state (e.g., two AC executions trying to add the same role at the same time) by running each AC execution in a separate and isolated context, as suggested in [123]. In detail, all AC executions of a workflow — and of other workflows as well — share the same initial state, but then any state-change rule — albeit being applied over the same state — is customized to each AC execution. Concretely, this means that identifiers of users, roles, and transient resources are made unique to each AC execution. For instance, two AC executions $c_1$ and $c_2$ of the workflow in Figure 3.1 induced by a basic RBAC system (see Section 3.4.1) would not both

create the "Clerk" role, but instead two "Clerk$_1$" and "Clerk$_2$" roles. If an organization deems that creating a different version of the same role for each AC execution of a workflow is not realistic or representative of the modeled business process, then the organization can just avoid running multiple

**Algorithm 7:** The pseudocode of the initializer

**Input:** $\Psi, \{C_1, ..., C_n\}, \langle |\mathbf{U}|, |\mathbf{R}|, |\mathbf{F}|, |\mathbf{UR}|, |\mathbf{PA}|, m_{(u \to r)}, M_{(u \to r)}, m_{(r \to u)}, M_{(r \to u)}, m_{(\langle f,op \rangle \to r)}, M_{(\langle f,op \rangle \to r)}, m_{(r \to \langle f,op \rangle)}, M_{(r \to \langle f,op \rangle)} \rangle$

**Output:** $\langle \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{UR}, \mathbf{PA} \rangle$

assert $(|\mathbf{U}| \cdot m_{(r \to u)}) \leq |\mathbf{R}| \leq (|\mathbf{U}| \cdot M_{(r \to u)}) \wedge (|\mathbf{R}| \cdot m_{(u \to r)}) \leq |\mathbf{U}| \leq (|\mathbf{R}| \cdot M_{(u \to r)}) \wedge (|\mathbf{R}| \cdot m_{(\langle f,op \rangle \to r)}) \leq |\mathbf{F}| \leq (|\mathbf{R}| \cdot M_{(\langle f,op \rangle \to r)}) \wedge (|\mathbf{F}| \cdot m_{(r \to \langle f,op \rangle)}) \leq |\mathbf{R}| \leq (|\mathbf{F}| \cdot M_{(r \to \langle f,op \rangle)})$;
let $\mathbf{U} = \mathbf{R} = \mathbf{F} = \mathbf{UR} = \mathbf{PA} = \emptyset$;
**if** *(addUser(·)* $\notin \Psi$*)* **then**
    let $u =$ `getUsernames`$(|\mathbf{U}|, \{C_1, ..., C_n\})$;
    **for** $i \leftarrow 1$ *to* $|\mathbf{U}|$ **do**
        add $u[i]$ to $\mathbf{U}$;
    **end**
**end**
**if** *(addRole(·)* $\notin \Psi$*)* **then**
    let $r =$ `getRoleNames`$(|\mathbf{R}|, \{C_1, ..., C_n\})$;
    **for** $i \leftarrow 1$ *to* $|\mathbf{R}|$ **do**
        add $r[i]$ to $\mathbf{R}$;
    **end**
**end**
let $f =$ `getPersistentResourceNames`$(|\mathbf{F}|, \{C_1, ..., C_n\})$;
**for** $i \leftarrow 1$ *to* $|\mathbf{F}|$ **do**
    add $f[i]$ to $\mathbf{F}$;
**end**
**if** *(assignUserToRole(·,·)* $\notin \Psi$*)* **then**
    let $u2r = \{\}$; `// Num of assigned roles per user`
    let $r2u = \{\}$; `// Num of assigned users per role`
    let *counter* $= 0$; `// Total user-role assignments`
    **foreach** *u in* $\mathbf{U}$ **do**
        **while** $u2r[u] < m_{(r \to u)}$ **do**
            pick $r \in \mathbf{R} \mid r2u[r] < M_{(u \to r)}$;
            invoke `assignUserToRole`$(u, r)$;
        **end**
    **end**
    **foreach** *r in* $\mathbf{R}$ **do**
        **while** $r2u[r] < m_{(u \to r)}$ **do**
            pick $u \in \mathbf{U} \mid u2r[u] < M_{(r \to u)}$;
            invoke `assignUserToRole`$(u, r)$;
        **end**
    **end**
    **while** *counter* $< |\mathbf{UR}|$ **do**
        pick $u \in \mathbf{U} \mid u2r[u] < M_{(r \to u)}$;
        pick $r \in \mathbf{R} \mid r2u[r] < M_{(u \to r)}$;
        invoke `assignUserToRole`$(u, r)$;
    **end**
**end**
`...  // Assign permissions to roles`

**Function** `getUsernames`$(m, \{C_1, ..., C_n\})$**:**
    `// return` $m$ `user names from` $\{C_1, ..., C_n\}$
**Function** `getRoleNames`$(m, \{C_1, ..., C_n\})$**:**
    `// return` $m$ `role names from` $\{C_1, ..., C_n\}$
**Function** `getPersistentResourceNames`$(m, \{C_1, ..., C_n\})$**:**
    `// return` $m$ `resource names from` $\{C_1, ..., C_n\}$
**Function** `assignUserToRole`$(u, r)$**:**
    add $(u, r)$ to $\mathbf{UR}$;
    set $u2r[u] = u2r[u] + 1$;
    set $r2u[r] = r2u[r] + 1$;
    set *counter* $=$ *counter* $+ 1$;

AC executions of the same workflow concurrently. Otherwise, the organization can assume the role to already exist in the initial state, thus choosing a highly or partially customized RBAC system (see Section 3.4.1). The same reasoning applies to users and transient resources, while persistent resources — according to their semantics — are shared among all AC executions.

Finally, whenever a user is selected to carry out an action (e.g., read a resource) through a role in an AC execution, that user is marked as *busy*, and no other AC execution — even of other workflows — can select that user until the user's first action is completed. As also suggested in [123], this approach concurs in simulating a realistic environment in which each user carries out a single activity at a given time, but can take part in more workflows simultaneously. If no user is available to carry out an action in an AC execution (because, e.g., all users are busy with other activities), the AC execution becomes *idling* until a user with the required role assignment is available. The idling time is measured separately from the workflow execution time and may potentially be used to identify bottlenecks in the AC policy, i.e., important roles with too few users assigned because of, e.g., separation of duty constraints; we defer the study of this possibility to future work (see Section 5.1.2).

### 3.4.3 Implementation of `ACME`

We implement `ACME` in Python and make the implementation available as open-source software.[6] `ACME` is independent of the underlying AC enforcement mechanism, that is, `ACME` has to be complemented with an *adapter* (i.e., a Python subclass) acting as an interface toward a specific AC enforcement mechanism. In other words, an adapter for `ACME` specifies how to translate a given state-change rule in Table 3.2 into the sequence of API invocations to implement that rule in the corresponding mechanism. We implement 3 adapters in `ACME` for OPA, the AuthzForce Server (Community Edition) open-source implementation of XACML, and `CryptoAC`. We implement the first two adapters by encoding RBAC policy states as described in the OPA documentation[7] and the XACML standard,[8] respectively, while `CryptoAC`'s APIs already have a 1-to-1 mapping with the state-change rules in Table 3.2.[9] We remark that the way a mechanism actually carries out a state-change rule (e.g., assigning permissions resource-wise, based on patterns, or by distributing cryptographic keys to authorized users) depends on the implementation of the mechanism itself, whose performance is exactly the object of our evaluation. In this regard, we notice that the performance of a mechanism is not limited to the evaluation of state-change rules only, but it also includes the handling of resources. For instance, CAC-based enforcement mechanisms perform cryptographic computations on resources [9]; omitting these computations would lead to non-realistic results. For this reason, we represent a resource as a 1MB file, although organizations can easily modify this representation (e.g., using a message queue instead of files for an IoT-based workflow) or the file size resource-wise in `ACME`. Then, we implement the adapters so that each of the 4 state-change rules concerning (both transient and persistent) resources — i.e., addResource($f$), deleteResource($f$), and the entailment relation for read and write queries over resources — considers the handling of resources themselves. For instance, reading a file in `CryptoAC` implies both evaluating the entailment relation and also downloading — and decrypting — the (1MB file representing the) resource. Further details can be found directly in the documentation of the adapters.[10]

[6] ACME — https://github.com/stfbk/ACME/

[7] Role-based access control (RBAC) (https://www.openpolicyagent.org/docs/latest/comparison-to-other-systems/)

[8] Index of /xacml/3.0/ (http://docs.oasis-open.org/xacml/3.0/)

[9] CryptoAC Documentation (https://cryptoac.readthedocs.io/en/latest/)

[10] ACME — https://github.com/stfbk/ACME/

Finally, ACME uses Locust,[11] a modern open-source load testing tool, as the underlying framework to invoke APIs while collecting two metrics, i.e., the number of Requests per Second (RPS) and the response time of each API invocation.

Locust can perform load testing through any kind of API, not only HTTP-based, and can easily distribute the load generation across multiple devices[12] to, e.g., test the scalability of a mechanism, as we show in Section 3.5.

## 3.5 Applications of ACE and ACME

In this section, we first define the experimental settings for the application of our methodology (Section 3.5.1). Then, we use our methodology to evaluate CryptoAC (in detail, the CAC RBAC scheme presented in Section 4.3), OPA, and the AuthzForce implementation of XACML (Section 3.5.2). Finally, we discuss the obtained results, arguing the internal and external validity of our evaluation (Section 3.5.3) and comparing them against micro-benchmarking (Section 3.5.4).[13]

### 3.5.1 Experimental Settings

We now define the set of workflows $\{W_1, ..., W_n\}$, RBAC system $\langle \gamma, \Psi \rangle$, and infrastructure used in our experimentation.

**Workflows and RBAC System.** Our experimental evaluation does not involve a specific scenario or organization, thus we lack both the business processes (i.e., the workflows) and the RBAC system. Therefore, we employ 5 workflows from the official OMG report [46], i.e., "The Pizza Collaboration" ($W_1$ hereafter) — already presented in Section 3.1.1 — the "The Nobel Prize" ($W_2$), the "Incident Management as Detailed Collaboration" ($W_3$), the "BPI Web Registration with Moderator" ($W_4$) and the "Patient Treatment - Collaboration" ($W_5$).[14] We choose these workflows among those available in the OMG report as they group activities under pools and lanes — allowing for clear identification of roles, as discussed in Section 3.3.4 — contain exclusive gateways — originating different sequences of state-change rules — and encompass several types of resources (i.e., messages, data objects, and data stores). Moreover, the use of CAC (i.e., CryptoAC) is relevant to workflows dealing with sensitive data (e.g., $W_2$, $W_5$) that need to be protected by both AC and cryptography. We parse these workflows with ACE (see Section 3.3), obtaining $\{C_1, ..., C_5\}$ and set all weights (recall the concept of weight in Section 3.4.2) to 1 for the sake of simplicity. ACE identifies 576 unique WF net executions among 5.586 for $W_2$, 313 unique WF net executions among 761 for $W_3$, 11 unique WF net executions among 18 for $W_4$ and 9 unique WF net executions among 9 for $W_5$. Combined, these 5 workflows expect 17 users, 17 roles, and 39 (4 persistent and 35 transient) resources. Finally, we partially customize the RBAC system (see Section 3.4.1) by considering that the AC policy state already contains users, roles, persistent resources, and assignments of users and permissions to roles (as it is usually the case [35]). Concretely, we obtain $\gamma$ from the initializer by providing as input $\Psi = \{\vdash (\langle \cdot \cdot \rangle)\}$, $\{C_1, ..., C_5\}$ and the *healthcare* state blueprint (shown in Table 3.3), whose numbers for roles and resources are closer to these workflows than those of the other blueprints.

**Infrastructure.** We present in Figure 3.6 the infrastructure used in our evaluation (we do not represent Locust or the database components for the sake of simplicity). For both configurations

[11] Locust (https://locust.io/)

[12] Distributed load generation — Locust 2.22.0 documentation (https://docs.locust.io/en/stable/running-distributed.html)

[13] The results, plots, and replication package are available at https://github.com/stfbk/ACME/tree/A_Simulation_Framework_Replication_Package

[14] The workflows are available at https://github.com/stfbk/ACME/tree/A_Simulation_Framework_Replication_Package

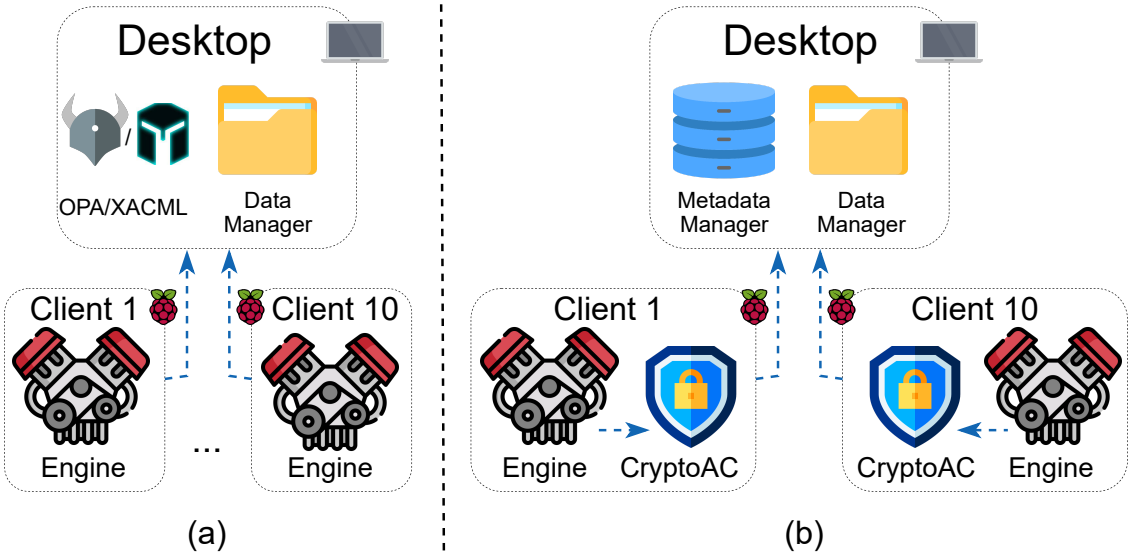


Figure 3.6: Infrastructure for (a) centralized and (b) decentralized Evaluations

(i.e., centralized with OPA or XACML and decentralized with `CryptoAC`), we use 10 Raspberry Pi 3 Model B+[15] running the Raspberry Pi Operative System (OS) as clients hosting the engine.

For the centralized configuration (Figure 3.6a), we deploy OPA or the AuthzForce Server for XACML on a desktop Dell Precision T1650[16] running Linux Mint 19 cinnamon. The deployment of OPA consists of two (micro) services, i.e., OPA itself (which evaluates state-change rules) and a data manager for handling resources created, updated, read, and deleted during the execution of workflows (recall the discussion about resources in Section 3.4.3); similarly, the deployment of XACML consists of the AuthzForce Server and the data manager. As said in Section 3.4.3, we encode the RBAC policy in OPA and in the AuthzForce Server as described in the OPA documentation[17] and the XACML standard,[18] respectively, and develop the data manager as a simple Kotlin program exposing four APIs following the CRUD paradigm to handle resources. For the decentralized configuration (Figure 3.6b), we deploy both the engine and `CryptoAC` on each of the Raspberry Pi. Indeed, `CryptoAC` provides decentralized RBAC policy enforcement through cryptography, i.e., `CryptoAC` is expected to run on the clients in a distributed fashion. The deployment of `CryptoAC` consists of three (micro) services: `CryptoAC` itself — which performs the cryptographic computations described in [9] — a data manager (we reuse the same service of the centralized configuration) and a metadata manager for metadata (e.g., public keys). The last two services run on the desktop centrally, as expected by [9].

We connect all devices (i.e., the 10 Raspberry Pi and the desktop) to the same wired LAN to guarantee a reliable network. We measure the latency and the throughput from each Raspberry Pi to the desktop with the `iperf3` tool,[19] finding them to be less than 1 ms and equal to 294 Mbps, respectively.

[15] Each Raspberry Pi is equipped with a Cortex-A53 at 1.4GHz with 1GB of LPDDR2 SDRAM and Gigabit ethernet with a maximum throughput of 300 Mbps

[16] The desktop is equipped with an Intel Xeon E3-1240 V2 at 3.40GHz with 16GB of DDR3 RAM and Gigabit Ethernet

[17] Role-based access control (RBAC) (https://www.openpolicyagent.org/docs/latest/comparison-to-other-systems/)

[18] Index of /xacml/3.0/ (http://docs.oasis-open.org/xacml/3.0/)

[19] iPerf (https://iperf.fr/iperf-download.php)

### 3.5.2 Experimental Evaluation

Below, we define how we elaborate the measurements collected during the evaluation, present the experiments we conducted, and report the obtained results.

**Measurements Elaboration.** We are interested in evaluating the performance — i.e., scalability, response time and throughput — of OPA, XACML and `CryptoAC`. Therefore, we aggregate individual response times of HTTPS requests to determine the overall average response time for the AC executions of a workflow — that is, the average overhead due to the AC enforcement mechanism in the execution of the workflow (*workflow response time*). Moreover, we measure the number of RPS (*throughput*). The amount of RAM consumed may be another relevant measurement. However, `CryptoAC` is deployed on different devices with respect to OPA and the AuthzForce Server, and consists of a different number of services (as described in Section 3.5.1). Thus, their RAM consumptions are not comparable, at least not directly. As such, we decide to not measure the amount of RAM consumed. Moreover, we highlight that there already exist several tools (e.g., `top`,[20] Valgrind[21]) allowing to measure the RAM consumed by a process.

**Experiments.** To investigate the (presumed) benefits of the decentralization of `CryptoAC` with respect to the (centralized) OPA and XACML (i.e., to measure scalability), we exploit Locust's distributed load generation capabilities (see Section 3.4.3) to scale the number of clients (i.e., engine instances) running AC executions concurrently from 5 to 10. In other words, we repeat the experiment 6 times for `CryptoAC`, OPA and XACML (i.e., 18 experiments in total), each time increasing the number of clients (thus, increasing the load on centralized services). At the beginning of each experiment, we launch the engine in each of the clients involved in the current evaluation, providing as input $\{C_1, ..., C_5\}$ and the RBAC system $\langle \gamma, \Psi \rangle$ (see Section 3.5.1). We activate clients 5 seconds apart from each other to avoid them starting at the same time; this allows intertwining AC executions of workflows, i.e., ensuring that AC executions run at different stages, as in a realistic scenario. We let the engine run AC executions multiple times by setting a time limit of 20 minutes. In short, we run each experiment separately, resetting all services each time.

**Results.** As clients activate 5 seconds apart from each other (i.e., with 10 clients, it takes 45 seconds to fully start the experiment), we discard the first minute's worth of results in which not all clients have started already; we do this for all 18 experiments to match their duration. We report in Table 3.7 the evaluation results, from 5 to 10 clients (*Cl*), considering only AC executions that run entirely in the experimentation time span. In detail, we report the average workflow response time of AC executions of the 5 workflows and the average throughput (*TR*) for OPA (columns 2—7), XACML (columns 8—13) and `CryptoAC` (columns 14—19). We also chart the results of Table 3.7 for $W_1$ in the plot in Figure 3.7:[22] on the X-axis we report the number of users, while on the Y-axis we report the average workflow response time (in milliseconds). We mark average values for OPA with blue circles, while we mark average values for XACML and `CryptoAC` with red crosses and green squares, respectively. Finally, the black line traversing each set of marks of a mechanism is the linear regression model computed on the average values. As Locust keeps track of the number of RPS globally, we highlight that the throughput amounts at the number of HTTPS requests satisfied (i.e., answered) per second by the mechanisms across AC executions of the 5 workflows. We remark once again that the average workflow response times shown in Figure 3.7 and reported in Table 3.7 do

[20] top(1) - Linux manual page (https://man7.org/linux/man-pages/man1/top.1.html)

[21] Valgrind (https://valgrind.org/)

[22] The plots for $W_2$, $W_3$, $W_4$, and $W_5$ are available at https://github.com/stfbk/ACME/tree/A_Simulation_Framework_Replication_Package

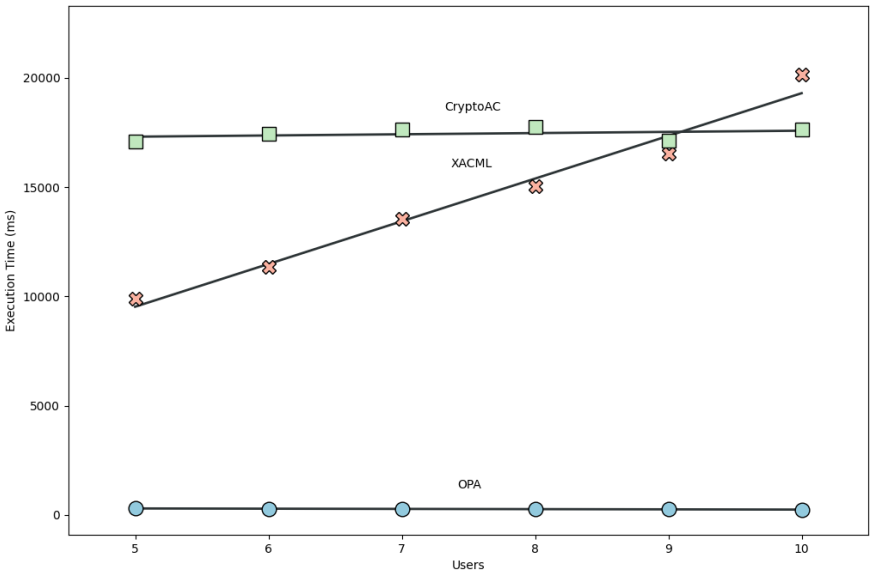


Figure 3.7: Average workflow response times for $W_1$

not refer to the time an AC enforcement mechanism takes to evaluate a single AC state-change rule, but rather to the evaluation of the whole workflow $W_1$ (which comprises 54 state-change rules, as explained in Section 3.3.5) when concurrently executed by 10 users — that is, when stressing the AC enforcement mechanism with many AC requests. For instance, if we consider the average throughput results with 10 clients reported in Table 3.7 (last row of columns 7, 13, and 19), we infer that OPA, XACML, and `CryptoAC` take 3.30ms, 59.38ms, and 35.63ms to elaborate a single AC state-change rule on average, respectively (that is, 1000ms divided by 302.9, 16.84, and 28.07).

Table 3.7: Average workflow response time (in ms) and throughput *TR* (i.e., number of requests per second)

| | | Number of Clients | | | | | |
|---|---|---|---|---|---|---|---|
| | | 5 | 6 | 7 | 8 | 9 | 10 |
| OPA | $W_1$ | 301.92 | 285.14 | 274.62 | 258.75 | 256.01 | 250.58 |
| | $W_2$ | 457.65 | 429.02 | 405.00 | 384.27 | 370.08 | 362.40 |
| | $W_3$ | 407.25 | 382.00 | 361.99 | 343.26 | 336.02 | 329.83 |
| | $W_4$ | 111.34 | 107.35 | 101.90 | 102.67 | 95.75 | 92.50 |
| | $W_5$ | 327.14 | 312.71 | 298.62 | 278.00 | 272.84 | 263.71 |
| | *TR* | 141.70 | 172.45 | 206.93 | 238.16 | 271.10 | 302.90 |
| XACML | $W_1$ | 9,880.10 | 11,339.37 | 13,556.74 | 15,032.91 | 16,552.14 | 20,138.11 |
| | $W_2$ | 15,059.98 | 17,282.57 | 20,474.54 | 24,043.45 | 25,708.71 | 31,046.99 |
| | $W_3$ | 9,792.55 | 11,836.76 | 13,086.97 | 14,486.40 | 15,996.81 | 17,419.01 |
| | $W_4$ | 3,191.33 | 3,626.97 | 3,940.23 | 4,398.52 | 5,258.16 | 6,116.17 |
| | $W_5$ | 11,210.61 | 13,958.79 | 15,584.98 | 16,523.94 | 19,245.75 | 20,815.66 |
| | *TR* | 16.39 | 16.49 | 17.01 | 17.13 | 17.37 | 16.84 |
| `CryptoAC` | $W_1$ | 17,091.03 | 17,460.31 | 17,624.60 | 17,749.26 | 17,113.88 | 17,656.92 |
| | $W_2$ | 26,577.29 | 27,027.49 | 27,275.60 | 27,265.83 | 25,398.23 | 26,038.53 |
| | $W_3$ | 16,185.51 | 16,141.53 | 15,933.43 | 16,136.32 | 15,806.56 | 16,035.27 |
| | $W_4$ | 6,089.74 | 6,240.69 | 6,307.76 | 6,343.48 | 6,179.73 | 6,263.67 |
| | $W_5$ | 18,227.53 | 18,309.22 | 18,500.20 | 18,745.12 | 18,119.65 | 18,505.22 |
| | *TR* | 14.02 | 16.70 | 19.31 | 21.91 | 25.69 | 28.07 |

### 3.5.3 Discussion

We can draw a number of considerations from Table 3.7 and Figure 3.7. First, we observe that OPA is faster and has a higher throughput than both XACML and `CryptoAC`. For instance, OPA takes 301.92ms on average for running an AC execution of $W_1$ with 5 clients, while XACML takes 9,880.10ms and `CryptoAC` takes 17,091.03ms. `CryptoAC` is the slowest among the evaluated mechanisms, probably because it enforces AC through cryptography, i.e., it performs several cryptographic computations for each state-change rule [9]. Instead, it is noticeable how the workflow response time of XACML — which is still a centralized mechanism — is an order of magnitude worse than the workflow response time of OPA. A possible explanation is that OPA stores all AC policies in cache memory, resulting in a faster evaluation; moreover, AC policies in OPA are stored as code instead of eXtensible Markup Language (XML), a cumbersome language to read and parse.[23] Then, XACML was designed to enforce ABAC rather than RBAC policies, and the standard encoding of RBAC for XACML seems to not be optimal. Finally, we remark that we are evaluating just one of the several implementations for XACML (i.e., the AuthzForce Server, which is not representative of all implementations of XACML) which is also open-source, hence not necessarily optimized or ready for production.

From Table 3.7 — and especially from Figure 3.7 — we also notice how the average workflow response times of AC executions change with the number of clients. The workflow response time of OPA across the 5 workflows remains almost constant from 5 to 10 clients, showing great vertical scalability and, perhaps, the presence of optimizations (e.g., caching); in any case, we highlight that these optimizations are the same that would be activated in a realistic scenario, as discussed in Section 3.2.3, thus they concur in producing realistic results. Conversely, the workflow response time of XACML quickly deteriorates when increasing the number of clients, suggesting low vertical scalability (e.g., from 9,880.1ms with 5 clients to 20,138.11ms with 10 clients for $W_1$, as shown in Figure 3.7); instead, the workflow response time of `CryptoAC` seems unaffected by the number of clients. In fact, the results indicate that `CryptoAC` has excellent vertical scalability — probably due to being a decentralized AC enforcement mechanism — even to the point that `CryptoAC` performs better than XACML when considering 10 clients, as illustrated in Figure 3.7. This impression is also confirmed by the results on the throughput, which increases linearly with the number of clients for `CryptoAC` (i.e., from 14.02 RPS with 5 clients to 28.07 RPS with 10 clients) while remains almost constant for XACML.

Finally, we remark that, besides performance, there exist other (subjective) conditions — not immediately related to performance but relevant nonetheless — which an organization may want to consider when choosing an AC enforcement mechanism, as discussed in Chapter 3. For instance, even though `CryptoAC` is slower than OPA, it provides greater security and privacy guarantees for the confidentiality and integrity of data through the use of cryptography. Thus, an organization should carefully evaluate the trade-off between performance and security, depending on the sensitivity of the involved data and the requirements of the underlying scenario.

**Internal validity.** As also argued in [16], internal validity is relatively easy to achieve, since we control all experimental settings (i.e., those discussed in Section 3.5.1) directly. In other words, it is possible to define the computational and network resources available, and even fine-tune them to better reproduce specific scenarios (e.g., by artificially introducing network latency with tools such as `tc`[24]). Then, the implementation of the core logic of the engine is common for all configurations (i.e., OPA, XACML and `CryptoAC`); only the adapter — which essentially just sends HTTPS requests

[23] OPA vs. XACML: Which Is Better for Authorization? - Styra (https://www.styra.com/blog/opa-vs-xacml-which-is-better-for-authorization/)

[24] tc(8) - Linux manual page (https://man7.org/linux/man-pages/man8/tc.8.html)

— differs. We even use the same data manager service in all configurations to avoid introducing biases. As a side note, we highlight that ACME is flexible enough to re-run the same evaluation with different components (e.g., data manager) to measure their efficiency in realistic scenarios. Finally, one may ask whether having both CryptoAC and the engine on the same device (i.e., the Raspberry Pi) could alter the results of the evaluation, as the two services might compete over the (limited) computational resources of the device. Therefore, during the experiments of CryptoAC, we monitor the CPU and RAM usage of the Raspberry Pi clients, finding them to be constantly around 40% and 30%, respectively. Consequently, we conclude that the two services can run together on the Raspberry Pi without any mutual interference, i.e., without altering the results.

**External validity** We now discuss whether our findings can generalize to actual deployments. First, we consider that often — although not always — workflows have a mix of activities carried out by software (i.e., automatically) and humans (i.e., manually) [123]. However, the results obtained with our methodology concern the overhead due to AC enforcement mechanisms in the execution of workflows only. To obtain more representative results, these can be integrated (i.e., summed) with the time needed by software agents and humans to carry out the activities in the workflows (which can be defined by organizations). Intuitively, this would dilute (in relative terms) eventual performance gaps among mechanisms. Nonetheless, this time is also independent of the mechanism being used, thus it does not alter its performance or a comparison with other mechanisms. Similarly, network latency — which is negligible in our evaluation — is another relevant aspect that may have a significant impact on the results of the experimentation. This is particularly true for the overall performance of centralized AC enforcement mechanisms (e.g., OPA and XACML) which suffer network latency more than decentralized mechanisms (e.g., CryptoAC) as, intuitively, the channel toward centralized services may become a bottleneck for the whole application. For instance, the latency for both AWS[25] and Microsoft Azure[26] Cloud — which naturally vary depending on the selected region and the location of the user — was found to be around 200ms on average in 2021 [89]. Suppose now that an organization deploys OPA — which mainly targets Cloud environments[27] — on AWS or Azure; considering that any AC execution of $W_1$ has 54 state-change rules — and considering one HTTPS request for each rule — only the overhead due to the latency of the Cloud would be more than ten seconds (i.e., 10,800ms $= 200\text{ms} \cdot 54$). Therefore, organizations should consider network latency, as well as all other network characteristics, to obtain results that are representative of a specific scenario.

Then, in our methodology we do not consider the impact that other processes may have in case they are carried out on the same device hosting the AC enforcement mechanism (e.g., CryptoAC), possibly leading to scarcity of (computational or network) resources. However, this behavior can be easily simulated by using dedicated tools (e.g., `stress`[28]).

Finally, as we do not consider a specific scenario for the sake of generality, we choose 5 BPMN workflows from the official report of OMG [46] to consider well-designed workflows only. Naturally, these workflows belong to different application fields. As a consequence, the results we obtain in our evaluation are still realistic (as they are built starting from a realistic RBAC state and sequences of state-change rules derived from BPMN workflows) but not representative of any concrete scenario. This issue is indeed solved when an organization uses its own workflows as input to our methodology.

### 3.5.4 Micro-benchmark Evaluation

We now compare the results of our methodology (presented in Section 3.5.2) against those of an evaluation based on micro-benchmarks. In this context, we consider the AuthzForce implementation of XACML and `CryptoAC` as AC enforcement mechanisms, since their performance is comparable while OPA outperforms both by far. In detail, we reuse the same experimental settings described in Section 3.5.1 for micro-benchmarking (e.g., RBAC system, infrastructure), while fixing the number of clients to 5. However, this time we consider a single state-change rule at a time. In other words, an experiment based on micro-benchmarking consists of 5 clients all sending the same state-change rule toward a mechanism for 10 minutes and then measuring the average response time, similarly to what we do in Section 3.5.2. As a side note, we highlight that this experimentation shows how it is possible to use our methodology also to run micro-benchmarks that consider a realistic RBAC system and have multiple clients sending state-change rules concurrently, two requirements to obtain significant results, as discussed in Section 3.2.3.

We report in Table 3.8 the results of the micro-benchmarks evaluation (best times are underlined). As we can see from the table, XACML performs better than `CryptoAC` in the majority of the considered state-change rules (i.e., 4 out of 6). In particular, XACML is faster at applying state-change rules, probably because `CryptoAC` involves the use of (slow) asymmetric cryptography for, e.g., distributing permissions. However, given that in CAC the enforcement of the policy is pre-compiled and embedded into the (distribution of the) secret keys, `CryptoAC` outperforms XACML when evaluating the entailment relation (i.e., read and write on resources). In other words, XACML has to evaluate policy sets and policies for each (user) AC request sent by all of the 5 clients, while in `CryptoAC` the evaluation is implicit. Looking at Table 3.8, an organization may then choose XACML over `CryptoAC` because generally better performing, but it is clear that the real answer actually depends on how frequent read and write actions are; this is an aspect that micro-benchmarks cannot capture, while our methodology can.

## 3.6 Related Work for Our Methodology

Below, we first review proposals in the literature to infer information concerning AC policies and AC requests from workflows modeling business processes (Section 3.6.1) — thus, related to `ACE`. Then, we investigate how performance evaluation is conducted in works presenting novel AC enforcement mechanisms and analyze the solutions proposed to evaluate the performance of such mechanisms (Section 3.6.2) — thus, related to `ACME`.

[25] AWS (https://aws.amazon.com/)
[26] Microsoft Azure (https://azure.microsoft.com/)
[27] Open Policy Agent (https://www.openpolicyagent.org/)
[28] impose load on/stress test systems - Linux man page (http://linux.die.net/man/1/stress)

Table 3.8: Micro-benchmarks average response time (ms)

| State-Change Rule | XACML | `CryptoAC` |
|---|---|---|
| addResource(·) (Transient Resource) | <u>12.75</u> | 615.43 |
| $\vdash (\langle \cdot, \langle$ `read` $\rangle\rangle)$ (Transient/Persistent Resource) | 3,132.32 | <u>884.94</u> |
| $\vdash (\langle \cdot, \langle$ `write` $\rangle\rangle)$ (Persistent Resource) | 3,568.01 | <u>861.31</u> |
| deleteResource(·) (Transient Resource) | <u>9.85</u> | 152.26 |
| assignPermissionToRole(·, ·) | <u>281.84</u> | 495.91 |
| revokePermissionFromRole(·, ·) | <u>175.74</u> | 270.38 |

### 3.6.1 Infer Access Control Information from Workflows

Domingos et al. [33] propose a technique to infer (parts of) RBAC policies from workflows represented as Unified Modeling Language (UML) activity diagrams. The authors identify roles by assuming the presence of UML supply objects defining what qualification (i.e., role) is required to perform a certain activity. Consequently, each role has permission to carry out the activity to which it is connected through the UML supply object. Regarding resources, the authors assume that all activities have generic input and output by default.

The authors in [128] propose a model to express RBAC policies in workflows. The model takes as input a workflow encoded as a partially ordered set of activities and a set of roles having permission to carry them out. Each activity (e.g., prepare a purchase order) already includes what information is required (e.g., amount of items and price per item) and what is produced (e.g., expense report). In other words, the authors assume to receive as input the roles and the resources, from which they infer the corresponding permissions.

In [38], the authors propose an enhanced RBAC model specific for business workflows. Their approach takes as input a workflow represented as a UML-like activity diagram, along with a textual description of the expected activities. The authors infer RBAC policies manually, extracting roles from the description (i.e., the semantics) of the activities. At the same time, they consider the activities themselves to be the resources over which to distribute permissions.

Also Uddin et al. [119] propose an enhanced RBAC model for workflows. As the focus of the authors is on the dynamicity of the authorizations and not on deriving AC information, they consider a single workflow and manually extract roles, permissions, and resources from the semantics of the activities. Then, they complement the RBAC policy by deciding what and how many users are present and the assignments of users to roles. The authors expect their policy to be enforced by a XACML-based enforcement mechanism.

Summarizing, all these approaches expect workflows to be represented either as UML activity diagrams or in a custom format, while in our work we address workflows in BPMN — the standard most widely adopted for modeling business processes [15, 104, 72]. Moreover, these approaches expect (part of) AC information to be either provided as straight input (i.e., [128]) or expressed through ad-hoc artifacts (e.g., UML supply objects in [33]) or embedded in the textual description of the activities [38, 119]. Instead, `ACE` infers information concerning AC policies and AC requests from workflows automatically based on their syntax and that of BPMN symbols. Moreover, `ACE` also enumerates all possible executions of workflows with the goal of benchmarking AC enforcement mechanisms.

### 3.6.2 Performance Evaluation of Access Control Mechanisms

Researchers usually measure the performance of AC enforcement mechanisms with micro-benchmarks which, however, are not representative of a realistic scenario, although some works propose techniques to evaluate (more or less generic) mechanisms. Below, we discuss the most relevant of these works.

**Micro-benchmarks.** In [102], the authors propose an AC enforcement mechanism based on XACML for smart energy grids to protect safety-relevant settings from either (unauthorized) malicious or (authorized) accidental changes. The authors measure the average response time of their mechanism for 1,000 requests to change safety-relevant settings with different combinations of CPUs and RAM, varying the AC policy by considering two sizes of attribute sets (10 and 100 attributes).

Martinelli et al. [76] propose a mechanism for OPC-UA-based industrial control applications allowing to define complex AC policies with continuous evaluation of AC requests, i.e., with the

possibility of stopping previously authorized actions when the policy conditions are not satisfied anymore. To measure the response time of their mechanism, the authors put a bound on the computational resources available, and then run experiments while varying the number of attributes in the policy.

In [9], the authors propose the `CryptoAC` tool to enforce RBAC policies through cryptography. The authors measure the response time of `CryptoAC` by invoking all `CryptoAC`'s APIs separately while ranging the number of roles, resources, and assigned permissions.

Ahmad et al. [2] investigate Cloud- and Edge-based ABAC enforcement mechanisms for distributed applications (e.g., smart home). Then, the authors propose a novel mechanism and evaluate its response time in AWS through micro-benchmarks, measuring the impact of Cloud vs. Edge deployment of the mechanism along with different attribute storage and retrieval strategies.

Summarizing, we find that the performance of AC enforcement mechanisms proposed in the literature is usually (either not measured or) evaluated through micro-benchmarks, i.e., by measuring the response time of single AC requests while varying specific elements of the AC policy like the number of roles (e.g., [9]) or attributes (e.g., [102, 76]). Moreover, the response time is often the only metric evaluated, while scalability and throughput are not considered. `ACME`, instead, aims at measuring the performance (i.e., scalability, response time, and throughput) of a mechanism throughout the intertwined execution of more instances of several workflows representative of the business processes of a specific scenario, thus considering the overall user experience from the users' point of view rather than focusing on single functionalities of the mechanism from the developers' point of view.

**Dedicated Techniques.** In [118], the authors evaluate the response time of 3 open-source XACML implementations with different combinations of XACML policies. In the experiments, the authors send single users' AC requests and measure the loading and evaluation time of the XACML implementations.

Ilhan et al. [56] study possible optimizations for caching in XACML. Similarly to [118], the authors measure the time for loading, filtering, and evaluating policies for a XACML-based mechanism of their choice. The authors consider 4 combinations of policies and measure the response time and the RAM consumption of the mechanism with single users' AC requests.

In [16] (and earlier work [17, 18, 45]), the authors present a framework, `ATLAS`, to measure the response time of AC enforcement mechanisms. First, they generate large numbers of XACML-based policies representative of a scenario, starting from a set of (manually) specified domain models and templates. Then, they evaluate the generated policies by sending AC requests derived from the policies themselves to XACML-based AC enforcement mechanisms. In `ATLAS`, each measurement is associated with predetermined settings, such as the policy set, the (single) users' AC request, and the computational resources available (i.e., CPU and RAM). In line with our thinking, the authors underline that performance evaluations must be based on realistic conditions to obtain significant results. However, the authors tackle this issue from the point of view of the policies (whose generation requires manual effort) but not of the realistic sequence of AC requests, as we do instead.

Similarly to [16], also the authors in [3] address the generation of realistic (XACML) policies. To this aim, they propose `XACBench`, a tool to generate synthetic XACML policies by extracting, modeling, and generalizing statistical properties of a given input policy. `XACBench` allows generating XACML policies of any size that model the characteristics of real-world policy sets.

Summarizing, the available literature for performance evaluation of AC enforcement mechanisms either targets specific standards (e.g., XACML in [118, 56]), thus restricting the applicability of the proposed approaches, or focus on the generation of realistic AC policies (e.g., in [16, 3]). Instead, the generation of realistic sequences of AC requests to measure the performance of generic mechanisms

is, to the best of our knowledge, still unaddressed. Finally, all the proposed approaches focus on the evaluation of standalone single users' AC requests (i.e., evaluating whether an AC request of a user to perform a particular action on a resource should be granted or denied), while ACME evaluates (sequences of) both users' and administrative AC requests.

# Chapter 4

# Design and Implementation of Two Cryptographic Access Control Schemes

As explained in Section 1.3, data produced, shared, and stored in cloud native applications are subject to an intricate threat model encompassing external attackers, malicious insiders, and even honest-but-curious cloud providers. In this context — also considering the limited trust on participating parties and the decentralized nature of cloud native applications — CAC is seen as a promising solution to provide both effective enforcement of AC policies and robust guarantees on data confidentiality and integrity (recall the discussion at the end of Section 1.4). Besides, we believe that CAC for cloud native applications should possess additional characteristics. In detail, the design of a CAC scheme should be compatible with the peculiarities of cloud native applications, and in particular with the microservice architectural design (see Section 1.1). Then, as said in Section 1.5, different scenarios may require supporting multiple AC models (e.g., RBAC, ABAC). Finally, a CAC scheme should be capable of protecting data in the Cloud-to-Thing Continuum (see Section 1.4). To the best of our knowledge, prior research has not addressed the *problem* of designing a security mechanism that can seamlessly fit into (the microservice-based architecture of) cloud native applications, support multiple AC models, and ensure E2E protection for data both in transit and at rest.

Therefore, below we address the aforementioned *problem* by presenting the design of two CAC schemes for RBAC and ABAC, respectively, and their implementation in a single CAC enforcement mechanism that possesses all of the aforementioned characteristics. First, after providing the necessary background notions (Section 4.1), we give an overview of the design of our CAC schemes (Section 4.2). Then, we present the CAC scheme for RBAC — whose design is inspired by the works in [39, 11] (Section 4.3) — and the CAC scheme for ABAC — which instead is a novel contribution of this thesis (Section 4.4). We describe the implementation of both CAC schemes into `CryptoAC`, a security mechanism suitable for use in cloud native applications (Section 4.5). Afterward, we use `CryptoAC` to conduct a thorough performance evaluation on the RBAC CAC scheme (Section 4.6). Finally, we compare our CAC schemes with related work (Section 4.7). We collect symbols used for the RBAC and ABAC CAC schemes in Tables 4.3 and 4.4, respectively.

## 4.1 Background and Preliminaries for Our Cryptographic Access Control Schemes

Below, we provide background information on RBAC and ABAC (Sections 4.1.1 and 4.1.2). Then, we describe the high-level functioning of ABE (Section 4.1.3). Finally, we briefly present the Message Queue Telemetry Transport (MQTT) protocol which we later employ in `CryptoAC`'s performance evaluation in Section 4.6 (Section 4.1.4).

### 4.1.1 Role-Based Access Control

RBAC is a relatively simple — at least with respect to ABAC (see Section 4.1.2) — AC model widely adopted in both academia and industry [19]. As mentioned in Section 3.1.3, in RBAC *users* are assigned to *roles* (e.g., job qualifications) which are assigned to *permissions*; *users* activate *roles* to access *permissions*. The state of a RBAC policy can be described as a tuple $\langle \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{UR}, \mathbf{PA} \rangle$, where $\mathbf{U}$ is the set of users, $\mathbf{R}$ is the set of roles, $\mathbf{F}$ is the set of resources, $\mathbf{UR} \subseteq \mathbf{U} \times \mathbf{R}$ is the set of user-role assignments and $\mathbf{PA} \subseteq \mathbf{R} \times \mathbf{PR}$ is the set of role-permission assignments, being $\mathbf{PR} \subseteq \mathbf{F} \times \mathbf{OP}$ a derivative set of $\mathbf{F}$ combined with a fixed set of actions $\mathbf{OP}$ (e.g., read, write). Both $\mathbf{OP}$ and $\mathbf{PR}$ are not part of the state, as $\mathbf{OP}$ remains constant over time and $\mathbf{PR}$ is derivative of $\mathbf{F}$ and $\mathbf{OP}$. We note that role hierarchies can always be compiled away by adding suitable pairs to $\mathbf{UR}$. A user $u$ can use a permission $\langle f, op \rangle$ if $\exists r \in \mathbf{R} \mid (u, r) \in \mathbf{UR} \wedge (r, \langle f, op \rangle) \in \mathbf{PA}$.

**Operations for Modifying an RBAC Policy State.** The set of operations allowing an administrator to modify the state of a RBAC policy is already presented in Section 3.1.3 — where the operations are called *state-change rules* — and reported in Table 3.2.

### 4.1.2 Attribute-Based Access Control

ABAC is a fine-grained AC model that regulates access to resources based on the evaluation of *attributes* relevant to making AC decisions [53]. In detail, ABAC may include three types of attributes:

- *subject attributes*, defining the characteristics of users (e.g., name, organization, role);
- *object attributes*, defining the characteristics of resources (e.g., title, size, category);
- *environmental attributes*, describing the context in which users' AC requests occur (e.g., date, network security level, timezone).

Attributes are also used as building blocks for *access structures*. Informally, an access structure implements a (rule of a) policy by specifying which subject, object, and environmental attributes are required to execute an operation over a specific resource. More formally, given a set of attributes $\mathbf{A}$, an access structure *as* is a subset of the power set of the set of attributes $\mathbf{A}$ minus the empty set, i.e., $as \subseteq (\mathcal{P}(\mathbf{A}) \setminus \emptyset)$. Each element in *as* is called *authorized attribute set* and it is said to *satisfy as*, whereas $(\mathcal{P}(\mathbf{A}) \setminus as)$ identifies the *unauthorized* attribute sets. For the purposes of our work, we consider subject attributes only hereafter — a choice which we explain in more detail at the end of Section 4.1.3. Hence, the state of an ABAC policy can be described as a tuple $\langle \mathbf{U}, \mathbf{A}, \mathbf{F}, \mathbf{UA}, \mathbf{FA} \rangle$, where $\mathbf{U}$ is the set of users, $\mathbf{A}$ is the set of (subject) attributes, $\mathbf{F}$ is the set of resources, $\mathbf{UA} \subseteq \mathbf{U} \times \mathbf{A}$ is the set of user-attribute assignments, and $\mathbf{FA} \subseteq \mathbf{PR} \times (\mathcal{P}(\mathbf{A}) \setminus \emptyset)$ is the set of permission-access structure assignments, being $\mathbf{PR} \subseteq \mathbf{F} \times \mathbf{OP}$ a derivative set of $\mathbf{F}$ combined with a fixed set of operations $\mathbf{OP}$ (e.g., read, write). Both $\mathbf{OP}$ and $\mathbf{PR}$ are not part of the state of the AC policy, as $\mathbf{OP}$ remains constant over time and $\mathbf{PR}$ is derivative of $\mathbf{F}$ and $\mathbf{OP}$. A user $u$ can use a permission $\langle f, op \rangle$ if $\exists (\langle f, op \rangle, \mathbf{A}') \in \mathbf{FA}\, s.t.\ \mathbf{A}' \subseteq \{a | (u, a) \in \mathbf{UA}\}$.

**Values for Subject Attributes.** Attributes in user-attribute assignments and in access structures may be enriched with a value represented as an arbitrary binary string $\{0,1\}^*$. In this case, the set of user-attribute assignments would be $\mathbf{UA} \subseteq \{(u, a, \mathbf{x}) | (u, a) \in (\mathbf{U} \times \mathbf{A}) \wedge \mathbf{x} \in \{0,1\}^*\}$; for instance, a user $u$ can be assigned to an attribute $a =$"*age*" with value $\mathbf{x} = 10010$ (i.e., 18), that is, $(u,$"*age*"$, 10010) \in \mathbf{UA}$. Similarly, an access structure *as* would be a subset of the power set of the set of attributes $\mathbf{A}$ minus the empty set plus an attribute value, i.e., $as \subseteq \mathcal{P}(\{(a, \mathbf{x}) | a \in \mathbf{A} \wedge \mathbf{x} \in$

$\{0,1\}^*\}) \setminus \emptyset$. In the current notation, we note that it is possible to assign to the same user $u$ the same attribute $a$ with different values — this possibility is not erroneous per se, as it depends on the semantics given by the administrator on the attribute itself. However, for simplicity, we assume that $(u,a,\mathbf{x}) \in \mathbf{UA} \wedge (u,a,\mathbf{x}') \in \mathbf{UA} \implies \mathbf{x} = \mathbf{x}'$. Similarly, $(a,\mathbf{x}) \in as \wedge (a,\mathbf{x}') \in as \implies \mathbf{x} = \mathbf{x}'$. As a final remark, we highlight that the possibility of expressing values for attributes — although greatly simplifying the management of ABAC policies — does not enhance the expressiveness of ABAC. In fact, values may also be concatenated with attributes without loss of generality; for instance, considering the previous example, $u$ could be assigned to an attribute "$age_{10010}$" — hence, $(u,$"$age_{10010}$"$)$. Intuitively, the latter may greatly complicate the definition of access structures.

**Operations for Modifying an ABAC Policy State.** Currently, there is no agreed upon standard model for ABAC [55, 111]. One of the most commonly adopted high-level descriptions of ABAC is provided by the NIST in [54]. However, the NIST does not specify the set of operations allowing an administrator to modify the state of an ABAC policy. Therefore, we define such a set of operations similarly to that of RBAC in Table 3.2. In particular, we expect two operations for each set composing the state of an ABAC policy for the model presented above, one for adding an element to the set and one for removing an element from the set. We report the set of operations in Table 4.1 (we remind the reader that our ABAC model considers subject attributes only, as explained in Section 4.1.3).

**Access Structures as Boolean Formulas.** For brevity and ease of use, access structures are usually compiled and expressed as boolean formulas over attributes [122]. For instance, consider a policy stating that the medical data on the electrocardiogram of a patient can be accessed only by either cardiologists or surgeons. Given a set of attributes $\mathbf{A} = \{cardiologist, nurse, surgeon\}$, the access structure $as$ implementing the policy can be expressed either as the set of authorized attribute sets $\{\{cardiologist\}, \{surgeon\}, \{nurse, surgeon\}, \ldots\}$ or, more concisely, as the boolean formula "$cardiologist \vee surgeon$". Compiling access structures into formulas allows easily expressing conditions over attributes and their values (e.g., using $<, \leq, \geq, >, =$ and $\neq$).

### 4.1.3 Attribute-Based Encryption

ABE — firstly introduced in [103] — is an asymmetric cryptosystem in which keys are built from ABAC attributes and access structures (see Section 4.1.2) [122]. Hence, ABE can naturally enforce ABAC policies cryptographically [53]. Indeed, unlike traditional asymmetric cryptosystems where

Table 4.1: The set of operations for modifying the state of an ABAC policy according to the ABAC model described in Section 4.1.2

| Operation | Resulting ABAC Policy State |
|---|---|
| addUser($u$) | $\langle \mathbf{U} \cup \{u\}, \mathbf{A}, \mathbf{F}, \mathbf{UA}, \mathbf{FA} \rangle$ |
| deleteUser($u$) | $\langle \mathbf{U} \setminus \{u\}, \mathbf{A}, \mathbf{F}, \mathbf{UA}, \mathbf{FA} \rangle$ |
| addAttribute($a$) | $\langle \mathbf{U}, \mathbf{A} \cup \{a\}, \mathbf{F}, \mathbf{UA}, \mathbf{FA} \rangle$ |
| deleteAttribute($a$) | $\langle \mathbf{U}, \mathbf{A} \setminus \{a\}, \mathbf{F}, \mathbf{UA}, \mathbf{FA} \rangle$ |
| addResource($f$) | $\langle \mathbf{U}, \mathbf{A}, \mathbf{F} \cup \{f\}, \mathbf{UA}, \mathbf{FA} \rangle$ |
| deleteResource($f$) | $\langle \mathbf{U}, \mathbf{A}, \mathbf{F} \setminus \{f\}, \mathbf{UA}, \mathbf{FA} \rangle$ |
| assignAttributeToUser($u,a,\mathbf{x}$) | $\langle \mathbf{U}, \mathbf{A}, \mathbf{F}, \mathbf{UA} \cup \{(u,a,\mathbf{x})\}, \mathbf{FA} \rangle$ |
| revokeAttributeFromUser($u,a$) | $\langle \mathbf{U}, \mathbf{A}, \mathbf{F}, \mathbf{UA} \setminus \{(u,a,-)\}, \mathbf{FA} \rangle$ |
| assignAccessStructureToResource($\langle f, op \rangle, as$) | $\langle \mathbf{U}, \mathbf{A}, \mathbf{F}, \mathbf{UA}, \mathbf{FA} \cup \{(\langle f, op \rangle, as)\} \rangle$ |
| revokeAccessStructurefromResource($\langle f, op \rangle$) | $\langle \mathbf{U}, \mathbf{A}, \mathbf{F}, \mathbf{UA}, \mathbf{FA} \setminus \{(\langle f, op \rangle, -)\} \rangle$ |
| $\gamma \vdash (\langle u, \langle f, op \rangle \rangle) : true \iff \exists (\langle f, op \rangle, as) \in \mathbf{FA}\ s.t.\ as \subseteq \{(a,\mathbf{x}) \mid (u,a,\mathbf{x}) \in \mathbf{UA}\}$ | |

there is a 1-to-1 mapping between public and private keys, ABE enables *broadcast encryption*, meaning that a single public key can be paired with multiple private keys. Consequently, it is possible to encrypt a resource — using a public ABE key — that can be decrypted by multiple authorized users — using their private ABE key.

In ABE, keys are generated by a central authority called Key Generation Authority (KGA), although variations are possible (e.g., multi-authority ABE [122]). Nonetheless, depending on how public and private ABE keys are built starting from attributes and access structures, we can identify two types of ABE cryptosystems, namely Key-Policy Attribute-Based Encryption (KP-ABE) [44] and Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [103, 12]. In KP-ABE, public keys embed attributes, while private keys embed access structures. Therefore, a user encrypting a resource (i.e., the data owner) decides what attributes are associated with that resource (i.e., object attributes) depending on what public key she uses. A recipient user (i.e., the data user) can decrypt a resource if the attributes of the resource satisfy the access structure — usually expressed as a boolean formula — of her private ABE key. As such, KP-ABE is suitable for implementing CAC schemes in which the policy may be unclear when resources are created, as in content-based AC [122]. Instead, in CP-ABE, public keys embed access structures, while private keys embed attributes. Therefore, a data owner decides what access structure (i.e., policy) to use to protect a resource, while data users can decrypt the resource only if they use a private ABE key embedding an authorized (subject) attribute set. As such, CP-ABE is more suitable than KP-ABE for implementing AC policies [12, 122]. For this reason, in the rest of the chapter, when mentioning ABE, we mean CP-ABE only.

As a final note, we highlight how (CP-)ABE enforces ABAC policies using subject attributes only. Indeed, attributes embedded in users' secret ABE keys can, intuitively, capture only users' (and not objects' or environmental) characteristics.

**Attribute-based Encryption Algorithms.** A generic ABE cryptosystem usually expects four algorithms:

- $(\mathbf{MPK}, \mathbf{MSK}) \leftarrow \mathbf{Setup}^{\mathbf{ABE}}(par)$: the setup algorithm — executed by the KGA — takes as input the security parameter $par$. The algorithm outputs the Master Public Key (MPK) **MPK** and Master Secret Key (MSK) **MSK**: the MPK is required to encrypt messages and decrypt ciphertexts, while the MSK allows the KGA to generate the users' private ABE keys;
- $\mathbf{sk} \leftarrow \mathbf{Gen}^{\mathbf{ABE}}(\mathbf{MPK}, \mathbf{MSK}, \mathbf{A}')$: the key generation algorithm — executed by the KGA — takes as input the master key pair $(\mathbf{MPK}, \mathbf{MSK})$ and a set of attributes $\mathbf{A}' \subseteq \mathbf{A}$. The algorithm outputs the private ABE key **sk** which embeds the attributes in $\mathbf{A}'$;
- $c \leftarrow \mathbf{Enc}^{\mathbf{ABE}}(\mathbf{MPK}, as, m)$: the encryption algorithm — executed by a data owner — takes as input **MPK**, an access structure $as$ and a plaintext $m$. The algorithm outputs a ciphertext $c$ encrypted under $as$;
- $m/\perp \leftarrow \mathbf{Dec}^{\mathbf{ABE}}(\mathbf{MPK}, \mathbf{sk}, c)$: the decryption algorithm — executed by a data user — takes as input **MPK**, a private ABE key **sk** and a ciphertext $c$. The algorithm either outputs the corresponding plaintext $m$ if the attributes in **sk** satisfy the access structure of $c$. Otherwise, the algorithms output an error symbol $\perp$.

### 4.1.4 The Message Queuing Telemetry Transport Protocol

MQTT is a lightweight topic-based publish-subscribe messaging protocol[1] widely employed in cloud native applications, especially (but not only) for IoT scenarios [88]. MQTT expects messages to be

[1] ISO/IEC 20922:2016 - Information technology — Message Queuing Telemetry Transport (MQTT) v3.1.1 (https://www.iso.org/standard/69466.html)

published to *topics*, which can be seen as (transient) communication channels grouping messages logically related to each other (e.g., concerning a specific location, event, or action). *MQTT clients* (e.g., IoT devices, microservices) can subscribe to a topic, thus expressing the will to receive messages published to that topic. Whenever a MQTT client wants to publish a message to a topic, it sends the message — together with the (case-sensitive UTF-8 encoded) name of the topic — to the *MQTT broker*, which can be seen as the central node of a star network topology. When the broker receives the message and the name of the topic, it broadcasts the message to all MQTT clients that previously subscribed to that topic.

## 4.2 Design Overview of the Cryptographic Access Control Schemes

Below, we describe high-level principles that guide the design of both our CAC schemes.

### 4.2.1 Assumptions

Since our focus is on the enforcement of AC policies through cryptography rather than on other aspects (still relevant but) less interesting research-wise (e.g., key management, correctness of roles and attributes associations with users), we design our CAC schemes by making 5 assumptions, which we report in Table 4.2. In detail, we assume public-private key pairs to be correctly associated with the corresponding users — or agents acting on behalf of users (e.g., laptops, smartphones) — noting that there exist several consolidated and well-known technical solutions (e.g., PKI) addressing this issue (*A1*). Then, we assume that the administrator is fully trusted to manage the AC policy (*A2*) and verify the associations of users with roles and attributes, e.g., with out-of-band communications (*A3*). Also, we assume that users securely manage their own private keys (*A4*), e.g., using tamper-proof hardware or a dedicated password-protected keystore [27], and that cryptographic primitives are perfect, i.e., the confidentiality and the integrity of ciphertexts cannot be compromised by attacking (e.g., with brute force or cryptanalysis) the cryptographic primitives themselves (*A5*).

### 4.2.2 Compliance With the Architectural Model

In Section 2.2, we identified the set of 4 entities on which CAC schemes are usually built, i.e., proxy, RM, MM, and DM (please refer to Section 2.2.2 for a detail description of each entity). Naturally, we base the design — and the following implementation — of our CAC schemes on these 4 entities. As already discussed in Section 2.5, such a design allows us to directly map CAC entities to microservices for cloud native applications.

### 4.2.3 Hybrid Encryption

Asymmetric cryptosystems — such as Rivest—Shamir—Adleman (RSA) [100], Elliptic-Curve Cryptography (ECC) [66, 80], and ABE [86] — are known to be computationally expensive and inadequate to encrypt large quantities of data. Consequently, as typically done in CAC (see Section 2.1),

Table 4.2: Our assumptions for designing our CAC schemes

| | |
|---|---|
| *A1* | Public keys are correctly associated with users' identities |
| *A2* | The administrator is fully trusted |
| *A3* | Roles and attributes are correctly associated with users |
| *A4* | Users securely manage their own private keys |
| *A5* | Cryptographic primitives are perfect |

we design our CAC schemes to use *hybrid cryptography*, i.e., the combination of asymmetric and symmetric cryptography [39]. In other words, we introduce a level of indirection in our CAC schemes by encrypting resources with symmetric keys which we then distribute to authorized users only with asymmetric cryptography. In particular, we consider the use of AEAD — which is a more robust and secure variant of authenticated encryption — to bind ciphertexts to the context where they are supposed to be used and avoid replay attacks.

### 4.2.4 Hybrid Enforcement

Reasonably, cloud native applications could involve resources which — according to their sensitivity — may not be worth the computational overhead of CAC. Moreover, we acknowledge that the use of cryptography alone makes it difficult — if possible at all — to evaluate users' AC requests depending on dynamic and contextual conditions (e.g., environmental attributes). Therefore, we consider the possibility of combining our CAC schemes with centralized AC enforcement mechanisms. Concretely, we design our CAC schemes by specifying how to implement in CAC each of the operations usually expected by administrators to modify the state of RBAC and ABAC policies (see Tables 3.2 and 4.1). In other words, we provide an interface on top of our CAC schemes — that, essentially, describes the available APIs — which is (or at least should be) compatible with that of centralized AC enforcement mechanisms. Then, in our CAC schemes we allow administrators to specify which kind of enforcement (either cryptographic, centralized, or both) to apply over a resource. By choosing each time the more appropriate enforcement, we can enhance the expressiveness of our CAC schemes and relieve their computational overhead. In other words, rather than supplanting centralized AC enforcement mechanisms, our CAC schemes can complement and synergize with them to provide an even more complete and thorough protection of sensitive resources. Nonetheless, note that centralized AC enforcement mechanisms may be bypassed by the (parties managing the) mechanisms themselves (e.g., cloud providers) — on these concerns, recall the discussion in Section 1.3.

### 4.2.5 Version Numbers for Secure Revocation

Intuitively, RBAC and ABAC policies evolve over time with the addition and removal of users, roles, attributes, and resources, as well as user-role, role-permission, user-attribute, and permission-access structure assignments. On this aspect, particular care should be put into the revoking of privileges to users, e.g., after the removal of a user-attribute assignment or the deletion of a role. In fact, one may think that renewing the user's ABE secret key without embedding the revoked attribute(s) or deleting all symmetric keys to which the deleted role could access may be enough to maintain a secure state. However, users may cache private or symmetric keys and — in case of privileges loss — use these cached keys for (unauthorized) access to resources, possibly colluding with honest-but-curious participating parties (e.g., cloud providers). Therefore, to maintain a secure state after the revoking of privileges to users, we design our CAC schemes to renew all affected cryptographic material by making old cached keys unusable. To this end, we assign version numbers to roles, attributes, and resources, linking the corresponding private and secret cryptographic keys to these version numbers. Whenever a role, attribute, or resource is involved in the revoking of privileges to a user, we increment the respective version number by one, accordingly renewing the affected private and secret cryptographic keys and distributing them to all other authorized users. However, the re-encryption of resources is a non-trivial matter that requires a dedicated discussion (see Section 4.2.6).

### 4.2.6 Re-Encryption of Resources

In CAC, whenever a user loses privileges and, as a consequence, cannot access a resource anymore, the corresponding symmetric key is renewed (recall the discussion in Section 4.2.5). Although new data of the resource are going to be encrypted with the new symmetric key, the already existing data of the resource can still be decrypted — and potentially accessed by the (now unauthorized) user — with the old symmetric key. In this circumstance, we identify four different courses of action:

- **immediate re-encryption**: resources can be immediately re-encrypted to prevent unauthorized users from accessing the data contained therein. This course of action may incur a large computational overhead and impact availability by locking access to a (possibly high) number of resources for a considerable amount of time, as each resource needs to be decrypted and re-encrypted;
- **no re-encryption**: resources are not re-encrypted to avoid the computational overhead and the loss of availability. In this course of action, we either rely on centralized AC enforcement mechanisms to prevent unauthorized access — possibly risking collusions — or we assume that unauthorized users already accessed the existing data before losing their privileges;
- **threshold-based re-encryption**: as a combination of the two previous courses of action, resources can be re-encrypted based on a threshold specified on the number of old symmetric keys. In other words, we parametrize the re-encryption of resources: immediate re-encryption and no re-encryption can be achieved by specifying a threshold of 1 or infinite, respectively. Moreover, administrators can specify threshold values in between for maximum flexibility.
- **lazy re-encryption**: resources are re-encrypted when the computational overhead and the loss of availability are less impactful, i.e., when the cloud native application is computationally unloaded (e.g., during nights) or whenever resources are overwritten. The latter means that, whenever an authorized user modifies a resource (e.g., a file) that has to be re-encrypted, that user simply re-encrypts the whole modified resource with the new symmetric key;

To explore alternative courses of action, our design employs lazy re-encryption in the RBAC CAC scheme (see Section 4.3) and threshold-based re-encryption in the ABAC CAC scheme (see Section 4.4).

### 4.2.7 Hiding Identifiers

Although resources are encrypted, participating parties (e.g., honest-but-curious cloud providers) may still infer sensitive information by observing the identifiers of users, roles, attributes, and resources (recall the discussion on metadata sensitivity in Section 2.2.4). Therefore, the design of our CAC schemes expects users, roles, attributes, and resources to be equipped with pseudonyms — that is, random sequences of bytes that are rotated periodically to maintain their effectiveness [93]. In detail, pseudonyms are used instead of identifiers to guarantee (or at least mitigate) indirect disclosure of sensitive information by the DM, the RM, and the proxy (recall the entities in architectural model for CAC in Section 2.2); the proxy of a user can only access those identifiers of roles, attributes, and resources the user is authorized for. Conversely, the MM — which manages the CAC policy — contains the association between the pseudonyms and the identifiers.

## 4.3 Role-Based Cryptographic Access Control

In this section, we propose a CAC scheme for cryptographically enforcing RBAC policies over sensitive data in transit and at rest in cloud native applications. The design of the RBAC CAC scheme

is based on the notions presented in Section 4.1.1, the design principles described in Section 4.2, and implements the operations reported in Table 3.2. We collect symbols used for the RBAC CAC scheme in Table 4.3.

### 4.3.1 Policy Encoding for Role-Based Cryptographic Access Control

As said in Section 4.1.1, the state of a generic RBAC policy consists of a tuple $\langle \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{UR}, \mathbf{PA} \rangle$. We encode an RBAC policy for CAC with (possibly decorated versions of) the same sets, as we explain below.

As common in CAC (see Section 2.1), each user $u$ is equipped with a public-private key pair for en/decryption $(\mathbf{k}_u^{\mathbf{enc}}, \mathbf{k}_u^{\mathbf{dec}}) \leftarrow \mathbf{Gen}^{\mathbf{P}}$ and a public-private key pair for signatures verifica-

Table 4.3: Symbols used in Section 4.3

| Symbol | Description | Symbol | Description |
|---|---|---|---|
| $adm$ | The administrator | $u$ | A user |
| $r$ | A role | $f$ | A resource |
| $fc$ | The content of the resource $f$ | $v$ | A version number |
| $m$ | A plaintext | $c$ | A ciphertext |
| $\mathbf{N}$ | Null (i.e., empty) value | $-$ | Wildcard |
| $\mathbf{p}$ | A pseudonym | $\mathbf{k}^{\mathbf{enc}}$ | A public encryption key |
| $\mathbf{k}^{\mathbf{dec}}$ | A private decryption key | $\mathbf{k}^{\mathbf{ver}}$ | A public verification key |
| $\mathbf{k}^{\mathbf{sig}}$ | A private signing key | $\mathbf{k}^{\mathbf{sym}}$ | A symmetric key |
| Read | The read action | Write | The write action |
| $op$ | A subset of the set of actions $\mathbf{OP}$ | $\mathbf{OP}$ | The set of actions |
| $\mathbf{Enc_c}$ | The cryptographic enforcement type | $\mathbf{Enc}$ | An enforcement type |
| $\mathbf{S1}$ | The flag for the incomplete status | $\mathbf{S2}$ | The flag for the operational status |
| $\mathbf{S3}$ | The flag for the deleted status | $\mathbf{S}$ | A flag for a status |
| $\mathbf{P_c}$ | The AC policy enforced traditionally | $\mathbf{P_t}$ | The AC policy enforced cryptographically |
| $\mathbf{P}$ | A generic AC policy | $\mathbf{PR}$ | The set of permissions |

| | |
|---|---|
| $\mathbf{Gen}^{\mathbf{P}}$ | The generation of a public-private key pair $(\mathbf{k}^{\mathbf{enc}}, \mathbf{k}^{\mathbf{dec}})$ for en/decryption |
| $\mathbf{Gen}^{\mathbf{Sig}}$ | The generation of a public-private key pair $(\mathbf{k}^{\mathbf{ver}}, \mathbf{k}^{\mathbf{sig}})$ for signatures verification/creation |
| $\mathbf{Gen}^{\mathbf{Pse}}$ | The generation of a pseudonym $\mathbf{p}$ |
| $\mathbf{Gen}^{\mathbf{Sym}}$ | The generation of a symmetric key $\mathbf{k}^{\mathbf{sym}}$ |
| $\mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}}$ | The asymmetric encryption computation with $\mathbf{k}^{\mathbf{enc}}$ |
| $\mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}}$ | The asymmetric decryption computation with $\mathbf{k}^{\mathbf{dec}}$ |
| $\mathbf{Enc}^{\mathbf{S}}_{\mathbf{k}^{\mathbf{sym}}}$ | The symmetric encryption computation with $\mathbf{k}^{\mathbf{sym}}$ |
| $\mathbf{Dec}^{\mathbf{S}}_{\mathbf{k}^{\mathbf{sym}}}$ | The symmetric decryption computation with $\mathbf{k}^{\mathbf{sym}}$ |
| $\langle \mathbf{U_t}, \mathbf{R_t}, \mathbf{F_t}, \mathbf{UR_t}, \mathbf{PA_t} \rangle$ | The state of the AC policy $\mathbf{P_t}$ enforced traditionally |
| $\mathbf{U_t}$ | The set of users in the AC policy $\mathbf{P_t}$; an element of the set has the form $u$ |
| $\mathbf{R_t}$ | The set of roles in the AC policy $\mathbf{P_t}$; an element of the set has the form $r$ |
| $\mathbf{F_t}$ | The set of resources in the AC policy $\mathbf{P_t}$; an element of the set has the form $f$ |
| $\mathbf{UR_t}$ | The set of user-role assignments in the AC policy $\mathbf{P_t}$; an element of the set has the form $(u, r)$ |
| $\mathbf{PA_t}$ | The set of role-permission assignments in the AC policy $\mathbf{P_t}$; an element of the set has the form $(r, \langle f, op \rangle)$ |
| $\langle \mathbf{U_c}, \mathbf{R_c}, \mathbf{F_c}, \mathbf{UR_c}, \mathbf{PA_c} \rangle$ | The state of the AC policy $\mathbf{P_c}$ enforced cryptographically |
| $\mathbf{U_c}$ | The set of users in the AC policy $\mathbf{P_c}$; an element of the set has the form $(u, \mathbf{p}_u, \mathbf{k}_u^{\mathbf{enc}}, \mathbf{k}_u^{\mathbf{ver}}, \mathbf{S})$ |
| $\mathbf{R_c}$ | The set of roles in the AC policy $\mathbf{P_c}$; an element of the set has the form $(r, \mathbf{p}_{(r,v_r)}, \mathbf{k}^{\mathbf{enc}}_{(r,v_r)}, \mathbf{k}^{\mathbf{ver}}_{(r,v_r)}, v_r, \mathbf{S})$ |
| $\mathbf{F_c}$ | The set of resources in the AC policy $\mathbf{P_c}$; an element of the set has the form $(f, \mathbf{p}_{(f,v_{f\_enc})}, v_{f\_dec}, v_{f\_enc}, \mathbf{S}, \mathbf{Enc})$ |
| $\mathbf{UR_c}$ | The set of user-role assignments in the AC policy $\mathbf{P_c}$; an element of the set has the form $(u, r, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_u}\left(\mathbf{k}^{\mathbf{enc}}_{(r,v_r)}\mathbf{k}^{\mathbf{dec}}_{(r,v_r)}, \mathbf{k}^{\mathbf{ver}}_{(r,v_r)}, \mathbf{k}^{\mathbf{sig}}_{(r,v_r)}\right), v_r)$ |
| $\mathbf{PA_c}$ | The set of role-permission assignments in the AC policy $\mathbf{P_c}$; an element of the set has the form $(r, f, \mathbf{p}_{(r,v_r)}, \mathbf{p}_{(f,v_f)}, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r)}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,v_{f\_dec})}\right), \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r)}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,v_{f\_enc})}\right) v_r, v_f, op)$ |

tion/creation $(\mathbf{k}_u^{\mathbf{ver}}, \mathbf{k}_u^{\mathbf{sig}}) \leftarrow \mathbf{Gen}^{\mathbf{Sig}}$. Similarly, each role $r$ with version number $v_r$ (recall the use of version numbers described in Section 4.2.5) is equipped with a public-private key pair for en/decryption $(\mathbf{k}_{(r,v_r)}^{\mathbf{enc}}, \mathbf{k}_{(r,v_r)}^{\mathbf{dec}}) \leftarrow \mathbf{Gen}^{\mathbf{P}}$ and a public-private key pair for signatures verification/creation $(\mathbf{k}_{(r,v_r)}^{\mathbf{ver}}, \mathbf{k}_{(r,v_r)}^{\mathbf{sig}}) \leftarrow \mathbf{Gen}^{\mathbf{Sig}}$. Each resource $f$ with version number $v_f$ is assigned to a symmetric key $\mathbf{k}_{(f,v_f)}^{\mathbf{sym}}$ to en/decrypt the content $fc$ of the resource $f$ itself. Resources can be created either by the administrator or by users; in the latter case, the user's operation to create a resource is mediated by the RM (see Section 2.2.2 for the description of the RM) to ensure the correctness and compliance of the user's operation with the policy. In both cases, a new symmetric key $\mathbf{k}_{(f,v_f)}^{\mathbf{sym}}$ is created (at first, $v_f = 1$). The design of the RBAC CAC schemes expects the administrator to have access to all resources. If the administrator wants to assign a user $u$ to a role $r$, the administrator encrypts $\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}$ and $\mathbf{k}_{(r,v_r)}^{\mathbf{sig}}$ with $\mathbf{k}_u^{\mathbf{enc}}$, resulting in $c = \mathbf{Enc}_{\mathbf{k}_u^{\mathbf{enc}}}^{\mathbf{P}}\left(\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}, \mathbf{k}_{(r,v_r)}^{\mathbf{sig}}\right)$ — in this way, $u$ can use $\mathbf{k}_u^{\mathbf{dec}}$ to access $r$'s private decryption and signing keys, i.e., $(\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}, \mathbf{k}_{(r,v_r)}^{\mathbf{sig}}) = \mathbf{Dec}_{\mathbf{k}_u^{\mathbf{dec}}}^{\mathbf{P}}(c)$ . If the administrator wants to assign a role $r$ a permission $\langle f, op \rangle$, the administrator encrypts $\mathbf{k}_{(f,v_f)}^{\mathbf{sym}}$ with $\mathbf{k}_{(r,v_r)}^{\mathbf{enc}}$, resulting in $c' = \mathbf{Enc}_{\mathbf{k}_{(r,v_r)}^{\mathbf{enc}}}^{\mathbf{P}}\left(\mathbf{k}_{(f,v_f)}^{\mathbf{sym}}\right)$ — in this way, a user $u$ that has access to $r$'s private decryption key $\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}$ can use it to access $f$'s symmetric key, i.e., $\mathbf{k}_{(f,v_f)}^{\mathbf{sym}} = \mathbf{Dec}_{\mathbf{k}_{(r,v_r)}^{\mathbf{dec}}}^{\mathbf{P}}(c')$ As mentioned in Section 4.2.7, users, roles, and resources are equipped with a pseudonym each to hide their identifiers. For instance, a resource can be referred to by its pseudonym rather than by its identifier, public keys are linked to users' and roles' pseudonyms, and users use their pseudonym when, e.g., creating digital signatures — if required. Finally, users, roles, and resources in $\mathbf{P_c}$ are assigned a status that can be either *incomplete* **S1** (i.e., created but not fully configured), *operational* **S2** (i.e., ready for use) or *deleted* **S3** (i.e., by the administrator).

Summarizing, the complete encoding of a RBAC policy for our RBAC CAC scheme is as follows (we underline fields which uniquely identify an element of the corresponding set):

$$(\underline{u}, \mathbf{p}_u, \mathbf{k}_u^{\mathbf{enc}}, \mathbf{k}_u^{\mathbf{ver}}, \mathbf{S}) \in \mathbf{U_c}$$
$$(\underline{r}, \mathbf{p}_{(r,v_r)}, \mathbf{k}_{(r,v_r)}^{\mathbf{enc}}, \mathbf{k}_{(r,v_r)}^{\mathbf{ver}}, v_r, \mathbf{S}) \in \mathbf{R_c}$$
$$(\underline{f}, \mathbf{p}_{(f,v_{f\_enc})}, v_{f\_dec}, v_{f\_enc}, \mathbf{S}, \mathbf{Enc}) \in \mathbf{F_c}$$
$$(\underline{u}, \underline{r}, \mathbf{Enc}_{\mathbf{k}_u^{\mathbf{enc}}}^{\mathbf{P}}\left(\mathbf{k}_{(r,v_r)}^{\mathbf{enc}}, \mathbf{k}_{(r,v_r)}^{\mathbf{dec}}, \mathbf{k}_{(r,v_r)}^{\mathbf{ver}}, \mathbf{k}_{(r,v_r)}^{\mathbf{sig}}\right), v_r) \in \mathbf{UR_c}$$
$$(\underline{r}, \underline{f}, \mathbf{p}_{(r,v_r)}, \mathbf{p}_{(f,v_f)}, \mathbf{Enc}_{\mathbf{k}_{(r,v_r)}^{\mathbf{enc}}}^{\mathbf{P}}\left(\mathbf{k}_{(f,v_{f\_dec})}^{\mathbf{sym}}\right), \mathbf{Enc}_{\mathbf{k}_{(r,v_r)}^{\mathbf{enc}}}^{\mathbf{P}}\left(\mathbf{k}_{(f,v_{f\_enc})}^{\mathbf{sym}}\right), v_r, \underline{v_f}, op) \in \mathbf{PA_c}$$

### 4.3.2 Full Construction for Role-Based Cryptographic Access Control

We report the pseudocode of our RBAC CAC scheme for administrative and users' AC requests in Figures 4.1 and 4.2, respectively. As anticipated in Section 4.2.4, we provide the pseudocode for all RBAC operations reported in Table 3.2, to which we add:

- $\underline{init_u()}$ and $\underline{init_{adm}()}$, for the initialization of a user $u$ and the administrator *adm* (i.e., generation of public-private key pairs and pseudonyms);
- $\underline{readResource_u(f)}$ and $\underline{writeResource_u(f, fc)}$ for read and write operations, respectively.

As said in Section 4.2.4, our RBAC CAC scheme allows for specifying which kind of enforcement **Enc** — either cryptographic $\mathbf{Enc_c}$, centralized $\mathbf{Enc_t}$, or both — to apply over a resource. As a result, we can identify two AC policy states, i.e., the one enforced cryptographically $\mathbf{P_c}$ and the one enforced traditionally $\mathbf{P_t}$. The administrator digitally signs both states to provide integrity and authenticity, while the integrity of data is guaranteed assuming the use of AEAD as the type for symmetric cryptosystem [6]; for the sake of simplicity, in Figures 4.1 and 4.2 we omit these details, as well as other trivial checks like the uniqueness of identifiers and pseudonyms.

$\underline{init_{adm}()}$
- Generate admin's public-private key pair for en/decryption $(\mathbf{k}^{\mathbf{enc}}_{adm}, \mathbf{k}^{\mathbf{dec}}_{adm}) \leftarrow \mathbf{Gen^P}$ and public-private key pair for signatures verification/creation $(\mathbf{k}^{\mathbf{ver}}_{adm}, \mathbf{k}^{\mathbf{sig}}_{adm}) \leftarrow \mathbf{Gen^{Sig}}$
- Add $(adm, adm, \mathbf{k}^{\mathbf{enc}}_{adm}, \mathbf{k}^{\mathbf{ver}}_{adm}, \mathbf{S2})$ to $\mathbf{U_c}$, $(adm, adm, \mathbf{k}^{\mathbf{enc}}_{adm}, \mathbf{k}^{\mathbf{ver}}_{adm}, 1, \mathbf{S2})$ to $\mathbf{R_c}$ and $(adm, adm, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{adm}}\left(\mathbf{k}^{\mathbf{enc}}_{adm}, \mathbf{k}^{\mathbf{dec}}_{adm}, \mathbf{k}^{\mathbf{ver}}_{adm}, \mathbf{k}^{\mathbf{sig}}_{adm}\right), 1)$ to $\mathbf{UR_c}$
- Add $adm$ to $\mathbf{U_t}$, $adm$ to $\mathbf{R_t}$ and $(adm, adm)$ to $\mathbf{UR_t}$

$\underline{addUser_{adm}(u)}$
- Add $(u, \mathbf{N}, \mathbf{N}, \mathbf{S1})$ to $\mathbf{U_c}$
- Add $u$ to $\mathbf{U_t}$

$\underline{deleteUser_{adm}(u)}$
- Replace $(u, -, -, -, -)$ with $(u, -, -, -, \mathbf{S3})$ in $\mathbf{U_c}$
- $\forall (r, -, -, -, -, -) \in \mathbf{R_c}\ s.t.\ (u, r, -, -) \in \mathbf{UR_c}$:
  - Invoke $\underline{revokeUserFromRole_{adm}(u, r)}$
- Delete $u$ from $\mathbf{U_t}$ and $(u, -)$ from $\mathbf{UR_t}$

$\underline{addRole_{adm}(r)}$
- Generate $r$'s public-private key pair for en/decryption $(\mathbf{k}^{\mathbf{enc}}_{(r,1)}, \mathbf{k}^{\mathbf{dec}}_{(r,1)}) \leftarrow \mathbf{Gen^P}$, public-private key pair for signatures verification/creation $(\mathbf{k}^{\mathbf{ver}}_{(r,1)}, \mathbf{k}^{\mathbf{sig}}_{(r,1)}) \leftarrow \mathbf{Gen^{Sig}}$ and pseudonym $\mathbf{p}_{(r,1)} \leftarrow \mathbf{Gen^{Pse}}$
- Add $(r, \mathbf{p}_{(r,1)}, \mathbf{k}^{\mathbf{enc}}_{(r,1)}, \mathbf{k}^{\mathbf{ver}}_{(r,1)}, 1, \mathbf{S2})$ to $\mathbf{R_c}$ and $(adm, r, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{adm}}\left(\mathbf{k}^{\mathbf{enc}}_{(r,1)}, \mathbf{k}^{\mathbf{dec}}_{(r,1)}, \mathbf{k}^{\mathbf{ver}}_{(r,1)}, \mathbf{k}^{\mathbf{sig}}_{(r,1)}\right), 1)$ to $\mathbf{UR_c}$
- Add $r$ to $\mathbf{R_t}$ and $(adm, r)$ to $\mathbf{UR_t}$

$\underline{deleteRole_{adm}(r)}$
- Delete all $(-, r, -, -)$ from $\mathbf{UR_c}$
- Replace $(r, -, -, -, -, -)$ with $(r, -, -, -, -, \mathbf{S3})$ in $\mathbf{R_c}$
- $\forall (f, -, -, -, -, -) \in \mathbf{F_c}\ s.t.\ (r, f, -, -, -, -, -, -, op) \in \mathbf{PA_c}$:
  - Invoke $\underline{revokePermissionFromRole_{adm}(r, \langle f, op \rangle)}$
- Delete $r$ from $\mathbf{R_t}$ and $(-, r)$ from $\mathbf{UR_t}$

$\underline{deleteResource_{adm}(f)}$
- Delete $(-, f, -, -, -, -, -, -, -)$ from $\mathbf{PA_c}$
- Replace $(f, -, -, -, -, -, -)$ with $(f, -, -, -, \mathbf{S3}, -)$ in $\mathbf{F_c}$
- Delete $f$ from $\mathbf{F_t}$ and $(-, \langle f, - \rangle)$ from $\mathbf{PA_t}$
- Ask the DM to delete $fc^{enc}$

$\underline{assignUserToRole_{adm}(u, r)}$
- Find $(adm, r, c, v_{(r,v_r)}) \in \mathbf{UR_c}$
- Decrypt $m = \mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_{adm}}(c)$
- Add $(u, r, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{u}}(m), v_{(r,v_r)})$ to $\mathbf{UR_c}$
- Add $(u, r)$ to $\mathbf{UR_t}$

$\underline{assignPermissionToRole_{adm}(r, \langle f, op \rangle)}$
- If $\exists\, op'\ s.t.\ (r, f, -, -, -, -, -, -, -, op') \in \mathbf{PA_c}$:
  - Replace $(r, f, -, -, -, -, -, -, -, op')$ with $(r, f, -, -, -, -, -, -, -, op \cup op')$ in $\mathbf{PA_c}$
  - Replace $(r, \langle f, op' \rangle)$ with $(r, \langle f, op \cup op' \rangle)$ in $\mathbf{PA_t}$
- Else:
  - Find $(adm, f, adm, \mathbf{p}_{(f,v_{f\_enc})}, kA_{dec}, kA_{enc}, 1, v_{f\_enc}, -) \in \mathbf{PA_c}$, $(r, -, \mathbf{k}^{\mathbf{enc}}_{(r,v_r)}, -, v_r, \mathbf{S2}) \in \mathbf{R_c}$ and $(f, \mathbf{p}_{(f,v_{f\_enc})}, v_{f\_dec}, v_{f\_enc}, \mathbf{S2}, \mathbf{Enc}) \in \mathbf{F_c}$
  - If $\mathbf{Enc} = \mathbf{Enc_c}$:
    - Decrypt $k_{dec} = \mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_{adm}}(kA_{dec})$ and $k_{enc} = \mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_{adm}}(kA_{enc})$
    - Set $kr_{dec} = \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r)}}(k_{dec})$ and $kr_{enc} = \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r)}}(k_{enc})$
  - Else:
    - Set $kr_{dec} = \mathbf{N}$ and $kr_{enc} = \mathbf{N}$
  - Add $(r, f, \mathbf{p}_{(r,v_r)}, \mathbf{p}_{(f,v_{f\_enc})}, kr_{dec}, kr_{enc}, v_r, v_{f\_enc}, op)$ to $\mathbf{PA_c}$
  - Add $(r, \langle f, op \rangle)$ to $\mathbf{PA_t}$

$\underline{revokeUserFromRole_{adm}(u, r)}$
- Find $(r, -, -, -, v_r, \mathbf{S2}) \in \mathbf{R_c}$
- Generate $r$'s new public-private key pair for en/decryption $(\mathbf{k}^{\mathbf{enc}}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{dec}}_{(r,v_r+1)}) \leftarrow \mathbf{Gen^P}$, new public-private key pair for signatures verification/creation $(\mathbf{k}^{\mathbf{ver}}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{sig}}_{(r,v_r+1)}) \leftarrow \mathbf{Gen^{Sig}}$, and new pseudonym $\mathbf{p}_{(r,v_r+1)} \leftarrow \mathbf{Gen^{Pse}}$
- Replace $(r, -, -, -, v_r, \mathbf{S2})$ with $(r, \mathbf{p}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{enc}}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{ver}}_{(r,v_r+1)}, v_r + 1, \mathbf{S2})$ in $\mathbf{R_c}$
- Find $(adm, r, c, v_{(r,v_r)}) \in \mathbf{UR_c}$
- Decrypt $\left(-, \mathbf{k}^{\mathbf{dec}}_{(r,v_r)}, -, -\right) = \mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_{adm}}(c)$
- For all $(u', r, -, -) \in \mathbf{UR_c}\ s.t.\ u' \neq u$:
  - Add $(u', r, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{u'}}\left(\mathbf{k}^{\mathbf{enc}}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{dec}}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{ver}}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{sig}}_{(r,v_r+1)}\right), v_r + 1)$ to $\mathbf{UR_c}$
- Delete all $(-, r, -, v_r)$ from $\mathbf{UR_c}$
- Delete $(u, r)$ from $\mathbf{UR_t}$
- $\forall (r, f, -, -, kr_{dec}, kr_{enc}, -, v_{f\_enc}, op) \in \mathbf{PA_c}$:
  - Find $(f, -, v_{f\_dec}, v_{f\_enc}, \mathbf{S2}, \mathbf{Enc}) \in \mathbf{F_c}$
  - If $\mathbf{Enc} = \mathbf{Enc_c}$:
    - Generate new symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,v_{f\_enc}+1)} \leftarrow \mathbf{Gen^{Sym}}$
    - Set $kr_{enc}{}' = \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r+1)}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,v_{f\_enc}+1)}\right)$
    - If $v_{f\_dec} = v_{f\_enc}$:
      - Set $kr_{dec}{}' = \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r+1)}}\left(\mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_{(r,v_r)}}(kr_{enc})\right)$
    - Else:
      - Set $kr_{dec}{}' = \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r+1)}}\left(\mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_{(r,v_r)}}(kr_{dec})\right)$
  - Else:
    - Set $\mathbf{k}^{\mathbf{sym}}_{(f,v_{f\_enc}+1)} = \mathbf{N}$
    - Set $kr_{enc}{}' = \mathbf{N}$ and $kr_{dec}{}' = \mathbf{N}$
  - Generate new pseudonym $\mathbf{p}_{(f,v_{f\_enc}+1)} \leftarrow \mathbf{Gen^{Pse}}$ for $f$
  - Replace $(f, -, v_{f\_dec}, v_{f\_enc}, -, -)$ with $(f, \mathbf{p}_{(f,v_{f\_enc}+1)}, v_{f\_dec}, v_{f\_enc} + 1, -, -)$ in $\mathbf{F_c}$
  - Add $(r, f, \mathbf{p}_{(r,v_r+1)}, \mathbf{p}_{(f,v_{f\_enc}+1)}, kr_{dec}{}', kr_{enc}{}', v_r + 1, v_{f\_enc} + 1, op)$ to $\mathbf{PA_c}$
  - For all $(r', f, \mathbf{p}_{(r',v_{r'})}, -, kr'_{dec}, kr'_{enc}, v_{r'}, -, op') \in \mathbf{PA_c}\ s.t.\ r' \neq r$:
    - Find $(r', -, \mathbf{k}^{\mathbf{enc}}_{(r',v_{r'})}, -, -, \mathbf{S2}) \in \mathbf{R_c}$
    - If $\mathbf{Enc} = \mathbf{Enc_c}$:
      - Set $kr'_{enc}{}' = \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r',v_{r'})}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,v_{f\_enc}+1)}\right)$
    - Else:
      - Set $kr'_{enc}{}' = \mathbf{N}$
    - If $v_{f\_dec} = v_{f\_enc}$:
      - Set $kr'_{dec}{}' = kr'_{enc}$
    - Else:
      - Set $kr'_{dec}{}' = kr'_{dec}$
    - Add $(r', f, \mathbf{p}_{(r',v_{r'})}, \mathbf{p}_{(f,v_{f\_enc}+1)}, kr'_{dec}{}', kr'_{enc}{}', v_{r'}, v_{f\_enc} + 1, op')$ to $\mathbf{PA_c}$
  - Delete all $(-, f, -, -, -, -, -, -, v_{f\_enc}, -)$ to $\mathbf{PA_c}$

$\underline{revokePermissionFromRole_{adm}(r, \langle f, op \rangle)}$
- Find $(r, f, -, -, -, -, -, -, op'') \in \mathbf{PA_c}$
- If $op'' \subseteq op$:
  - Find $(f, -, v_{f\_dec}, v_{f\_enc}, \mathbf{S2}, \mathbf{Enc}) \in \mathbf{F_c}$
  - If $\mathbf{Enc} = \mathbf{Enc_c}$:
    - Generate new symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,v_{f\_enc}+1)} \leftarrow \mathbf{Gen^{Sym}}$
  - Else:
    - Set $\mathbf{k}^{\mathbf{sym}}_{(f,v_{f\_enc}+1)} = \mathbf{N}$
  - Generate new pseudonym $\mathbf{p}_{(f,v_{f\_enc}+1)} \leftarrow \mathbf{Gen^{Pse}}$ for $f$
  - For all $(r', f, -, -, c, c', -, v_{f\_enc}, op') \in \mathbf{PA_c}\ s.t.\ r' \neq r$:
    - Find $(r', -, \mathbf{k}^{\mathbf{enc}}_{(r',v_{r'})}, -, -, \mathbf{S2}) \in \mathbf{R_c}$
    - If $\mathbf{Enc} = \mathbf{Enc_c}$:
      - Set $c'' = \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r',v_{r'})}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,v_{f\_enc}+1)}\right)$
    - Else:
      - Set $c'' = \mathbf{N}$
    - If $v_{f\_dec} = v_{f\_enc}$:
      - Set $c''' = c'$
    - Else:
      - Set $c''' = c$
    - Add $(r', f, -, \mathbf{p}_{(f,v_{f\_enc}+1)}, c''', c'', -, v_{f\_enc} + 1, op')$ to $\mathbf{PA_c}$
  - Delete all $(-, f, -, -, -, -, -, -, v_{f\_enc}, -)$ to $\mathbf{PA_c}$
  - Delete $(r, \langle f, op' \rangle)$ from $\mathbf{PA_t}$
  - Replace $(f, -, v_{f\_dec}, v_{f\_enc}, \mathbf{S2}, -)$ with $(f, \mathbf{p}_{(f,v_{f\_enc}+1)}, v_{f\_dec}, v_{f\_enc} + 1, \mathbf{S2}, -)$ in $\mathbf{F_c}$
- Else if $op \cap op'' \neq \emptyset$:
  - Replace $(r, f, -, -, -, -, -, -, -, op'')$ with $(r, f, -, -, -, -, -, -, -, op'' \setminus op)$ in $\mathbf{PA_c}$
  - Replace $(r, \langle f, op'' \rangle)$ with $(r, \langle f, op'' \setminus op \rangle)$ in $\mathbf{PA_t}$

Figure 4.1: Pseudocode for the administrative operations of the CAC scheme for RBAC

As a final remark, we highlight that in our RBAC CAC scheme, accountability — which we define as the ability to map ciphertexts to the users that created them — is currently not ensured cryptographically, since ciphertexts are encrypted with symmetric keys known by all authorized users. If necessary, users can be required to digitally sign ciphertexts using their private signing keys (or those of the role they assume), at the cost of increasing the computational overhead at the

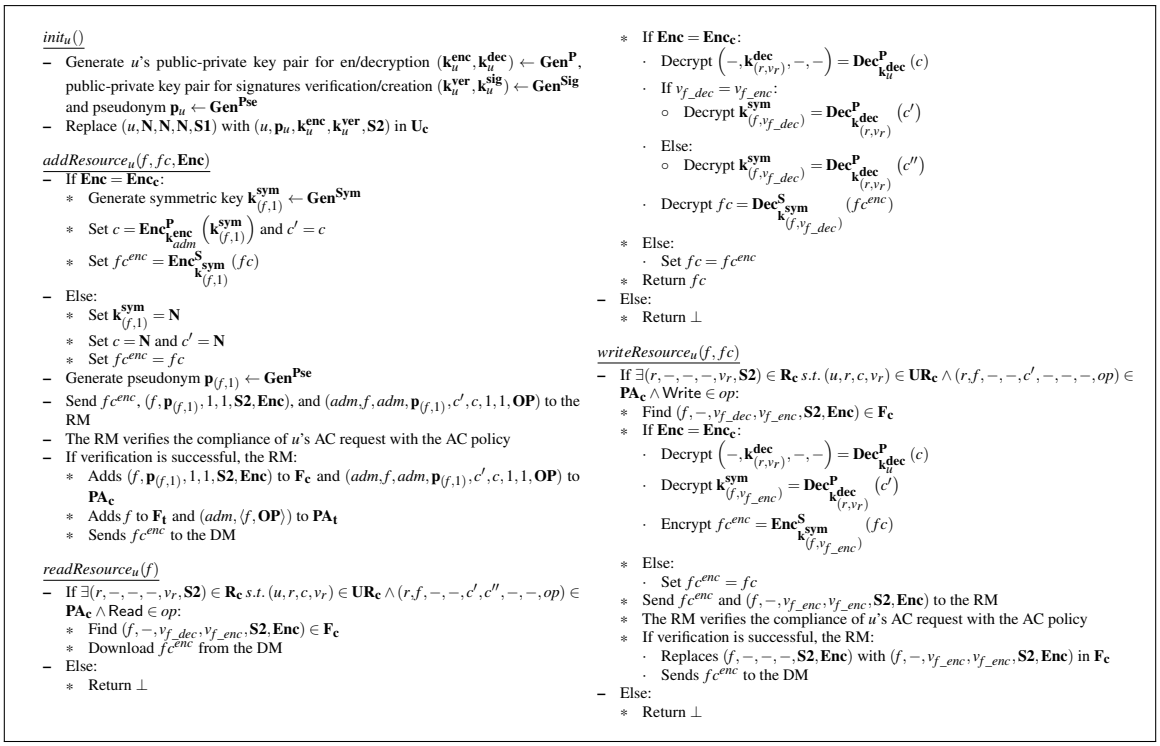


Figure 4.2: Pseudocode for the user operations of the CAC scheme for RBAC

client-side — the modification to the RBAC CAC scheme for implementing this behavior would be straightforward.

**Consistency Between $\mathbf{P_c}$ and $\mathbf{P_t}$.** Here, we focus on the key relationship between $\mathbf{P_c}$ and $\mathbf{P_t}$, which we can express as a set of invariants on the operations in Figures 4.1 and 4.2. Formally, we define a predicate *pre* as a boolean-valued function on the set of states of $\mathbf{P_c}$ or $\mathbf{P_t}$. The value of a predicate *pre* on a generic state $\mathbf{P}$ is denoted with $\mathbf{P} \models pre$. An operation $\psi$ — which, as explained at the end of Section 4.1.1, corresponds to a state-change rule — leaves a predicate *pre* invariant if and only if $\mathbf{P} \models pre$ implies $\mathbf{P'} \models pre$ whenever $\mathbf{P} \mapsto_{\psi} \mathbf{P'}$ (recall the notation from Section 3.1.3). In detail, any operation in Figures 4.1 and 4.2 leaves the following predicates invariant:

- $(u, -, -, -, -) \in \mathbf{U_c} \iff u \in \mathbf{U_t}$;
- $(r, -, -, -, -, -) \in \mathbf{R_c} \iff r \in \mathbf{R_t}$;
- $(f, -, -, -, -, -) \in \mathbf{F_c} \iff f \in \mathbf{F_t}$;
- $(u, r, -, -) \in \mathbf{UR_c} \iff (u, r) \in \mathbf{UR_t}$;
- $(r, f, -, -, -, -, -, -, op) \in \mathbf{PA_c} \iff (r, \langle f, op\rangle) \in \mathbf{PA_t}$.

By using the invariants above and recalling that — given a state $\langle \mathbf{U_t}, \mathbf{R_t}, \mathbf{F_t}, \mathbf{UR_t}, \mathbf{PA_t}\rangle$ — a user $u$ can use a permission $\langle f, op\rangle \iff \exists r \in \mathbf{R_t} \mid (u, r) \in \mathbf{UR_t} \wedge (r, \langle f, op\rangle) \in \mathbf{PA_t}$, it is easy to see that — given a state $\langle \mathbf{U_c}, \mathbf{R_c}, \mathbf{F_c}, \mathbf{UR_c}, \mathbf{PA_c}\rangle$ — the user $u$ can use the permission $\langle f, op\rangle \iff \exists (r, -, \mathbf{k}^{\mathbf{enc}}_{(r,v_r)}, -, -, -) \in \mathbf{R_c} \mid (u, r, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}_u^{\mathbf{enc}}}(-, \mathbf{k}^{\mathbf{dec}}_{(r,v_r)}, -, -), v_r) \in \mathbf{UR_c}$ and $(r, f, -, , \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r)}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,v_{f\_dec})}\right), \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r)}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,v_{f\_enc})}\right), v_r, v_f, op) \in \mathbf{PA_c}$. The invariants above show that $\mathbf{P_c}$ and $\mathbf{P_t}$ are synchronized on the authorization conditions depending on a core RBAC model. This allows refining such conditions with additional ones depending on contextual information (e.g., time-based permissions), as discussed in Section 4.2.4, that can be checked only with a centralized AC enforcement mechanism.

**Example of Execution for the $\underline{deleteUser_{adm}}$ Operation.** Many operations of our RBAC CAC scheme are easily understandable given the discussion in Section 4.2. To illustrate some of the more complex facets of the RBAC CAC scheme, here we describe how to delete a user $u$, an operation which involves (1) the removal of $u$'s information, (2) the revocation of $u$'s roles and (3) the update of involved resources and the re-keying of resources previously accessible by $u$:

1. we remove $u$ from $\mathbf{U_t}$ and mark $u$'s entry $(u, -, \mathbf{k}_u^{\mathbf{enc}}, \mathbf{k}_u^{\mathbf{ver}}, \mathbf{S2})$ as "deleted" (**S3**) in $\mathbf{U_c}$; we do not remove the entry as we may need $\mathbf{k}_u^{\mathbf{ver}}$ to verify digital signatures, and also to ensure that usernames are unique (see the $\underline{addUser_{adm}(u)}$ operation in Figure 4.1);

2. we identify the set of roles assigned to $u$, i.e.,$\{r|(r,-,-,-,-,-) \in \mathbf{R_c} \wedge (u,r,-,v_r) \in \mathbf{UR_c}\})$. For each identified role $r$, we revoke $r$ from $u$ by deleting the $(u,r)$ assignment from $\mathbf{UR_t}$ and the $(u,r,-,-)$ assignment from $\mathbf{UR_c}$. Then, we generate a new public-private key pair for en/decryption $(\mathbf{k}^{\mathbf{enc}}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{dec}}_{(r,v_r+1)}) \leftarrow \mathbf{Gen^P}$ and a new public-private key pair for signatures verification/creation $(\mathbf{k}^{\mathbf{ver}}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{sig}}_{(r,v_r+1)}) \leftarrow \mathbf{Gen^{Sig}}$ for $r$, increasing $v_r$ by 1 and generating a new pseudonym $\mathbf{p}_{(r,v_r+1)} \Leftarrow \mathbf{Gen^{Pse}}$. Then, we update $r$'s information — the new keys, pseudonym and version number — in $\mathbf{R_c}$ and distribute the new $r$'s keys to all *other* authorized users through new user-role assignments in $\mathbf{UR_c}$;

3. we identify the set of resources to which the set of roles previously identified have access, i.e.,$\{f|f \in \mathbf{F} \wedge (u,r,-,v_r) \in \mathbf{UR_c}\})$. For each identified resource $f$, we generate a new symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f+1)} \leftarrow \mathbf{Gen^{Sym}}$, increasing $v_f$ by 1 and generating a new pseudonym $\mathbf{p}_{(f,v_f+1)} \Leftarrow \mathbf{Gen^{Pse}}$. Then, we update $f$'s information — the new symmetric key, pseudonym, and version number — in $\mathbf{F_c}$ and distribute the new $f$'s key to all authorized roles through new role-permission assignments in $\mathbf{PA_c}$.

As a final remark, we highlight that deleted users, roles, and resources cannot be restored. This is just an arbitrary choice and we note that, if required by the underlying scenario, it is trivial to support both behaviors (i.e., uniqueness of names and restorability) for users, roles, and resources.

## 4.4 Attribute-Based Cryptographic Access Control

In this section, we propose a CAC scheme for cryptographically enforcing ABAC policies over sensitive data in transit and at rest in cloud native applications. The design of the CAC scheme is based on the notions presented in Section 4.1.2, the design principles described in Section 4.2, and implements the operations reported in Table 4.1. We collect symbols used for the ABAC CAC scheme in Table 4.4.

### 4.4.1 Policy Encoding for Attribute-Based Cryptographic Access Control

As said in Section 4.1.2, the state of a generic ABAC policy consists of a tuple $\langle \mathbf{U}, \mathbf{A}, \mathbf{F}, \mathbf{UA}, \mathbf{FA} \rangle$. We encode an ABAC policy for CAC with (possibly decorated versions of) the same sets, as we explain below.

As common in CAC (see Section 2.1), each user $u$ is equipped with a public-private key pair for en/decryption $(\mathbf{k}_u^{\mathbf{enc}}, \mathbf{k}_u^{\mathbf{dec}}) \leftarrow \mathbf{Gen^P}$ and a public-private key pair for signatures verification/creation $(\mathbf{k}_u^{\mathbf{ver}}, \mathbf{k}_u^{\mathbf{sig}}) \leftarrow \mathbf{Gen^{Sig}}$. Moreover, for each user $u$, the administrator generates an ABE secret key $\mathbf{sk}_u \leftarrow \mathbf{Gen^{ABE}}(\mathbf{MPK}, \mathbf{MSK}, \mathbf{A}_u)$ embedding the set of $u$'s attributes $\mathbf{A}_u \subseteq \mathbf{A}$. Then, the administrator encrypts $\mathbf{sk}_u$ with $u$'s encryption public key $encryptionKey_u$, resulting in $c = \mathbf{Enc}^{\mathbf{P}}_{encryptionKey_u}(\mathbf{sk}_u)$ — in this way, $u$ can use $\mathbf{k}_u^{\mathbf{dec}}$ to access the ABE private key, i.e., $\mathbf{sk}_u = \mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}_u^{\mathbf{dec}}}(c)$ Each resource $f$ with version number $v_f$ is assigned to a symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$ to en/decrypt the content $fc$ of the

Table 4.4: Symbols used in Section 4.4

| Symbol | Description | Symbol | Description |
|---|---|---|---|
| $adm$ | The administrator | $u$ | A user |
| $a$ | An attribute | $adm\_att$ | The admin attribute |
| $\mathbf{x}$ | An attribute value | $f$ | A resource |
| $fc$ | The content of the resource $f$ | $v$ | A version number |
| $m$ | A plaintext | $c$ | A ciphertext |
| $\mathbf{p}$ | A pseudonym | $t$ | A threshold number |
| $\mathbf{N}$ | Null (i.e., empty) value | $-$ | Wildcard |
| $\perp$ | Error | $l$ | A set of $(a,\mathbf{x})$ pairs |
| $as$ | An access structure | $\mathbf{sk}$ | An ABE secret key |
| $acc$ | A subset of $\mathbf{FA_c}$ | $\mathbf{k^{enc}}$ | A public encryption key |
| $\mathbf{k^{dec}}$ | A private decryption key | $\mathbf{k^{ver}}$ | A public verification key |
| $\mathbf{k^{sig}}$ | A private signing key | $\mathbf{k^{sym}}$ | A symmetric key |
| **MPK** | The ABE MPK | **MSK** | The ABE MSK |
| Read | The read action | Write | The write action |
| $op$ | A subset of the set of actions **OP** | **OP** | The set of actions |
| **S1** | The flag for the incomplete status | **S2** | The flag for the operational status |
| **S3** | The flag for the deleted status | **S** | A flag for a status |
| $\mathbf{Enc_t}$ | The traditional enforcement type | $\mathbf{Enc_c}$ | The cryptographic enforcement type |
| **Enc** | Either $\{\mathbf{Enc_t}\}$, $\{\mathbf{Enc_c}\}$, or $\{\mathbf{Enc_t},\mathbf{Enc_c}\}$ | $par$ | The security parameter |
| $\mathbf{P_c}$ | The AC policy enforced traditionally | $\mathbf{P_t}$ | The AC policy enforced cryptographically |
| **P** | A generic AC policy | **PR** | The set of permissions |

| | |
|---|---|
| $\mathbf{Setup^{ABE}}(par)$ | The generation of **MPK** and **MSK** from the security parameter $par$ |
| $\mathbf{Gen^{P}}$ | The generation of a public-private key pair $(\mathbf{k^{enc}},\mathbf{k^{dec}})$ for en/decryption |
| $\mathbf{Gen^{Sig}}$ | The generation of a public-private key pair $(\mathbf{k^{ver}},\mathbf{k^{sig}})$ for signatures verification/creation |
| $\mathbf{Gen^{Pse}}$ | The generation of a pseudonym $\mathbf{p}$ |
| $\mathbf{Gen^{Sym}}$ | The generation of a symmetric key $\mathbf{k^{sym}}$ |
| $\mathbf{Gen^{ABE}}(\mathbf{MPK},\mathbf{MSK},\mathbf{A}')$ | The generation of an ABE secret key **sk** embedding the set of attributes $\mathbf{A}' \subseteq \mathbf{A}$ |
| $\mathbf{Enc^{P}_{k^{enc}}}$ | The asymmetric encryption computation with $\mathbf{k^{enc}}$ |
| $\mathbf{Dec^{P}_{k^{dec}}}$ | The asymmetric decryption computation with $\mathbf{k^{dec}}$ |
| $\mathbf{Enc^{S}_{k^{sym}}}$ | The symmetric encryption computation with $\mathbf{k^{sym}}$ |
| $\mathbf{Dec^{S}_{k^{sym}}}$ | The symmetric decryption computation with $\mathbf{k^{sym}}$ |
| $\mathbf{Enc^{ABE}}(\mathbf{MPK},as,m)$ | The ABE encryption of the plaintext $m$ under the access structure $as$ |
| $\mathbf{Dec^{ABE}}(\mathbf{MPK},\mathbf{sk},c)$ | The ABE decryption of the ciphertext $c$ with the ABE secret key **sk** |
| $\mathbf{sat_1}(\mathbf{sk},as)$ | $\mathbf{sat_1}(\mathbf{sk},as)=1$ if the ABE secret key **sk** satisfies the access structure $as$, else 0 |
| $\mathbf{sat_2}(l,as)$ | $\mathbf{sat_2}(l,as)=1$ if the set of $(a,\mathbf{x})$ pairs $l$ satisfies the access structure $as$, else 0 |
| $\langle \mathbf{U_t},\mathbf{A_t},\mathbf{F_t},\mathbf{UA_t},\mathbf{FA_t}\rangle$ | The state of the AC policy $\mathbf{P_t}$ enforced traditionally |
| $\mathbf{U_t}$ | The set of users in the AC policy $\mathbf{P_t}$; an element of the set has the form $u$ |
| $\mathbf{A_t}$ | The set of attributes in the AC policy $\mathbf{P_t}$; an element of the set has the form $a$ |
| $\mathbf{F_t}$ | The set of resources in the AC policy $\mathbf{P_t}$; an element of the set has the form $f$ |
| $\mathbf{UA_t}$ | The set of user-attribute assignments in the AC policy $\mathbf{P_t}$; an element of the set has the form $(u,a,\mathbf{x})$ |
| $\mathbf{FA_t}$ | The set of permission-access structure assignments in the AC policy $\mathbf{P_t}$; an element of the set has the form $(\langle f,op\rangle,as)$ |
| $\langle \mathbf{U_c},\mathbf{A_c},\mathbf{F_c},\mathbf{UA_c},\mathbf{FA_c}\rangle$ | The state of the AC policy $\mathbf{P_c}$ enforced cryptographically |
| $\mathbf{U_c}$ | The set of users in the AC policy $\mathbf{P_c}$; an element of the set has the form $(u,\mathbf{p}_u,\mathbf{k}_u^{\mathbf{enc}},\mathbf{k}_u^{\mathbf{ver}},\mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}_u^{\mathbf{enc}}}(\mathbf{sk}_u),\mathbf{S})$ |
| $\mathbf{A_c}$ | The set of attributes in the AC policy $\mathbf{P_c}$; an element of the set has the form $(a,\mathbf{p}_{(a,v_a)},v_a,\mathbf{S})$ |
| $\mathbf{F_c}$ | The set of resources in the AC policy $\mathbf{P_c}$; an element of the set has the form $(f,\mathbf{p}_{(f,v_f)},v_f,t_f,\mathbf{Enc},\mathbf{S})$ |
| $\mathbf{UA_c}$ | The set of user-attribute assignments in the AC policy $\mathbf{P_c}$; an element of the set has the form $(u,a,\mathbf{x})$ |
| $\mathbf{FA_c}$ | The set of permission-access structure assignments in the AC policy $\mathbf{P_c}$; an element of the set has the form $(f,\mathbf{p}_{(f,v_f)},as,op,\mathbf{Enc^{ABE}}(\mathbf{MPK},as,\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}),v_f)$ |

resource $f$ itself. As for the RBAC CAC scheme in Section 4.3.1, resources can be created either by the administrator or by users. In both cases, a new symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$ is created (at first, $v_f = 1$). If the administrator wants to assign an attribute $a$ to a user $u$, the administrator re-generates $u$'s ABE secret key $abeSecretKey_u \leftarrow \mathbf{Gen}^{\mathbf{ABE}}(\mathbf{MPK}, \mathbf{MSK}, \mathbf{A}_u \cup \{a\})$ and re-distribute it to $u$ as described above. If the administrator wants to assign an access structure $as$ to a permission $\langle f, op \rangle$, the administrator encrypts $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$ under $as$, resulting in $c' = \mathbf{Enc}^{\mathbf{ABE}}(\mathbf{MPK}, as, \mathbf{k}^{\mathbf{sym}}_{(f,v_f)})$ — in this way, a user $u$ assigned to an authorized attribute set (recall the definition in Section 4.1.2) can use $abeSecretKey_u$ to access $f$'s symmetric key, i.e., $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)} = \mathbf{Dec}^{\mathbf{ABE}}(\mathbf{MPK}, \mathbf{sk}_u, c')$ As mentioned in Section 4.2.7, users, attributes, and resources are equipped with a pseudonym each to hide their identifiers. Finally, users, attributes, and resources in $\mathbf{P_c}$ are assigned a status that can be either *incomplete* **S1**, *operational* **S2** or *deleted* **S3**.

Summarizing, the complete encoding of an ABAC policy for our ABAC CAC scheme is as follows (we underline fields which uniquely identify an element of the corresponding set):

$$(\underline{u}, \mathbf{p}_u, \mathbf{k}^{\mathbf{enc}}_u, \mathbf{k}^{\mathbf{ver}}_u, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_u}(\mathbf{sk}_u), \mathbf{S}) \in \mathbf{U_c}$$
$$(\underline{a}, \mathbf{p}_{(a,v_a)}, v_a, \mathbf{S}) \in \mathbf{A_c}$$
$$(\underline{f}, \mathbf{p}_{(f,v_f)}, v_f, t_f, \mathbf{Enc}, \mathbf{S}) \in \mathbf{F_c}$$
$$(\underline{u}, \underline{a}, \mathbf{x}) \in \mathbf{UA_c}$$
$$(\underline{f}, \mathbf{p}_{(f,v_f)}, as, \underline{op}, \mathbf{Enc}^{\mathbf{ABE}}(\mathbf{MPK}, as, \mathbf{k}^{\mathbf{sym}}_{(f,v_f)}), \underline{v_f}) \in \mathbf{FA_c}$$

We note that, with this encoding, there may exist multiple access structure assignments for the same resource which differ in the operation assigned but share the same symmetric key. One may question whether having the same key for different operations is meaningful, i.e., how we can enforce distinct permissions if the key remains the same. In this regard, we remark that the symmetric key is encrypted each time under a *different* access structure, thus addressing a different subset of users, and that CAC can cryptographically enforce read actions only anyway [39], while the enforcement of other actions (e.g., write) requires the approval of the RM. Nonetheless, it is certainly possible to enrich this encoding by assigning different keys (or, e.g., authorization tokens) to different permissions. As a final remark, the administrator — being the KGA — can access all ABE ciphertexts by generating a suitable ABE secret key; this is required, e.g., when updating the state of the CAC policy and refreshing keys. However, a more usable alternative — which we adopt in our scheme — is to create a special attribute (e.g., *adm_att*) representing the admin status and add it in logical disjunction to all access structures.

### 4.4.2 Full Construction for Attribute-Based Cryptographic Access Control

We report the pseudocode of our ABAC CAC scheme for administrative and users' AC requests in Figures 4.3 and 4.4, respectively. As anticipated in Section 4.2.4, we provide the pseudocode for all ABAC operations reported in Table 4.1, to which we add:

- $\underline{init_u()}$ and $\underline{init_{adm}()}$, for the initialization of a user $u$ and the administrator *adm* (i.e., generation of public-private key pairs and pseudonyms);
- $\underline{keyGen_{adm}(u)}$ for the administrator *adm* to generate the user $u$'s ABE private key $\mathbf{sk}_u$;
- $\underline{updateAccessStructureOfResource_{adm}(f, acc)}$ collecting instructions for updating the permission-access structure assignments of a resource $f$;
- $\underline{readResource_u(f, v_f)}$ and $\underline{writeResource_u(f, fc)}$ for read and write operations, respectively.

$\underline{init_{adm}()}$
- Generate ABE master key pair $(\mathbf{MPK}, \mathbf{MSK}) \leftarrow \mathbf{Setup^{ABE}}(par)$ and make $\mathbf{MPK}$ public
- Generate admin's public-private key pair for en/decryption $(\mathbf{k}^{\mathbf{enc}}_{adm}, \mathbf{k}^{\mathbf{dec}}_{adm}) \leftarrow \mathbf{Gen^P}$ and public-private key pair for signatures verification/creation $(\mathbf{k}^{\mathbf{ver}}_{adm}, \mathbf{k}^{\mathbf{sig}}_{adm}) \leftarrow \mathbf{Gen^{Sig}}$
- Set $attributes = \emptyset$
- Add $adm\_att$ concatenated with version number 1 in $attributes$
- Generate $\mathbf{sk}_{adm} \leftarrow \mathbf{Gen^{ABE}}(\mathbf{MPK}, \mathbf{MSK}, attributes)$
- Add $(adm, adm, \mathbf{k}^{\mathbf{enc}}_{adm}, \mathbf{k}^{\mathbf{ver}}_{adm}, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{adm}}(\mathbf{sk}_{adm}), \mathbf{S2})$ to $\mathbf{U_c}$, $(adm\_att, adm\_att, 1, \mathbf{S2})$ to $\mathbf{A_c}$ and $(adm, adm\_att, \mathbf{N})$ in $\mathbf{UA_c}$
- Add $adm$ to $\mathbf{U_t}$, $adm\_att$ to $\mathbf{A_t}$ and $(adm, adm\_att, \mathbf{N})$ in $\mathbf{UA_t}$

$\underline{keyGen_{adm}(u)}$
- Get the ABE master key pair $\mathbf{MPK}, \mathbf{MSK}$
- Set $attributes = \emptyset$
- For all $(u, a, \mathbf{x}) \in \mathbf{UA_c}$:
  - Find $(a, -, v_a, \mathbf{S2}) \in \mathbf{A_c}$
  - Add $a$ concatenated with $v_a$ and value $\mathbf{x}$ in $attributes$
- Generate $\mathbf{sk}_u \leftarrow \mathbf{Gen^{ABE}}(\mathbf{MPK}, \mathbf{MSK}, attributes)$
- Find $(u, -, \mathbf{k}^{\mathbf{enc}}_u, -, -, \mathbf{S2})$ in $\mathbf{U_c}$
- Replace $(u, -, \mathbf{k}^{\mathbf{enc}}_u, -, -, \mathbf{S2})$ with $(u, -, \mathbf{k}^{\mathbf{enc}}_u, -, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_u}(\mathbf{sk}_u), \mathbf{S2})$ in $\mathbf{U_c}$

$\underline{addUser_{adm}(u)}$
- If $(u, -, -, -, -, -) \in \mathbf{U_c}$
  - $\perp$
- Add $(u, \mathbf{N}, \mathbf{N}, \mathbf{N}, \mathbf{N}, \mathbf{S1})$ to $\mathbf{U_c}$
- Add $u$ to $\mathbf{U_t}$

$\underline{deleteUser_{adm}(u)}$
- Set $l = \emptyset$
- For all $(a, -, v_a, \mathbf{S2}) \in \mathbf{A_c}$ *s.t.* $\exists (u, a, -) \in \mathbf{UA_c}$:
  - Add $(a, \mathbf{N})$ to $l$
- If $l \neq \emptyset$:
  - Invoke $\underline{revokeAttributeFromUser_{adm}(u, l)}$
- Replace $(u, -, -, -, -, -)$ with $(u, -, -, -, \mathbf{N}, \mathbf{S3})$ in $\mathbf{U_c}$
- Delete $u$ from $\mathbf{U_t}$

$\underline{addAttribute_{adm}(a)}$
- If $(a, -, v_a, \mathbf{S}) \in \mathbf{A_c}$:
  - If $\mathbf{S} = \mathbf{S3}$:
    - Generate new pseudonym $\mathbf{p}_{(a, v_a+1)} \leftarrow \mathbf{Gen^{Pse}}$
    - Replace $(a, -, v_a, \mathbf{S3})$ with $(a, \mathbf{p}_{(a, v_a+1)}, v_a + 1, \mathbf{S2})$ in $\mathbf{A_c}$
  - Else:
    - $\perp$
- Else:
  - Generate new pseudonym $\mathbf{p}_{(a,1)} \leftarrow \mathbf{Gen^{Pse}}$
  - Add $(a, \mathbf{p}_{(a,1)}, 1, \mathbf{S2})$ to $\mathbf{A_c}$
- Add $a$ to $\mathbf{A_t}$
- Invoke $\underline{assignAttributeToUser_{adm}(adm, \{(a, \mathbf{N})\})}$

$\underline{deleteAttribute_{adm}(l)}$
- Set $users = \emptyset$
- For all $(a, -) \in l$:
  - For all $(u, -, -, -, -, \mathbf{S2}) \in \mathbf{U_c}$ *s.t.* $(u, a, -) \in \mathbf{UA_c}$:
    - Add $u$ to $users$
  - Delete $(-, a, -)$ from $\mathbf{UA_c}$
  - Delete $(-, a, -)$ from $\mathbf{UA_t}$
  - Replace $(a, -, -, \mathbf{S2})$ with $(a, -, -, \mathbf{S3})$ in $\mathbf{A_c}$
  - Delete $a$ from $\mathbf{A_t}$
- For all $u \in users$:
  - Invoke $\underline{keyGen_{adm}(u)}$
- For all $(f, -, -, -, -, -, \mathbf{S2}) \in \mathbf{F_c}$ *s.t.* $\exists (f, -, as, -, -, v_f) \in \mathbf{FA_c} \wedge (l \cap as) \neq \emptyset$:
  - Set $acc = \emptyset$
  - For all $(f, -, as', op, c, v'_f) \in \mathbf{FA_c}$:
    - Set $as'' = as' \setminus l$
    - Add $(f, -, as'', op, c, v'_f)$ to $acc$
  - Invoke $\underline{updateAccessStructureOfResource_{adm}(f, acc)}$

$\underline{deleteResource_{adm}(f)}$
- Replace $(f, -, -, -, -, \mathbf{S2})$ with $(f, -, -, -, -, \mathbf{S3})$ in $\mathbf{F_c}$
- Delete all $(f, -, -, -, -, -)$ from $\mathbf{FA_c}$
- Delete $f$ from $\mathbf{F_t}$ and $(\langle f, - \rangle, -)$ from $\mathbf{FA_t}$
- Delete all $(f, fc, -)$ from the DM

$\underline{assignAttributeToUser_{adm}(u, l)}$
- For all $(a, \mathbf{x}) \in l$:
  - If $(a, -, -, \mathbf{S3}) \in \mathbf{A_c} \vee \nexists (a, -, -, -) \in \mathbf{A_c}$ :
    - $\perp$
  - Add $(u, a, \mathbf{x})$ in $\mathbf{UA_c}$
  - Add $(u, a, \mathbf{x})$ in $\mathbf{UA_t}$
- Invoke $\underline{keyGen_{adm}(u)}$

$\underline{revokeAttributeFromUser_{adm}(u, l)}$
- Set $users = \emptyset$
- For all $(a, -) \in l$ *s.t.* $(a, -, v_a, \mathbf{S2}) \in \mathbf{A_c} \wedge (u, a, -) \in \mathbf{UA_c}$:
  - For all $(u', -, -, -, -, \mathbf{S2}) \in \mathbf{U_c}$ *s.t.* $(u', a, -) \in \mathbf{UA_c}$:
    - Add $u'$ to $users$
  - Delete $(u, a, -)$ from $\mathbf{UA_c}$
  - Delete $(u, a, -)$ from $\mathbf{UA_t}$
  - Generate new pseudonym $\mathbf{p}_{(a, v_a+1)} \leftarrow \mathbf{Gen^{Pse}}$
  - Replace $(a, -, v_a, \mathbf{S2})$ with $(a, \mathbf{p}_{(a, v_a+1)}, v_a + 1, \mathbf{S2})$ in $\mathbf{A_c}$
- For all $u'' \in users$:
  - Invoke $\underline{keyGen_{adm}(u'')}$
- For all $(f, -, -, -, -, \mathbf{S2}) \in \mathbf{F_c}$ *s.t.* $\exists (f, -, as, -, -, v_f) \in \mathbf{FA_c} \wedge (l \cap as) \neq \emptyset$:
  - Set $acc = \emptyset$
  - For all $(f, -, as', op, c, v'_f) \in \mathbf{FA_c}$:
    - For all $(a', -) \in l$ *s.t.* $a' \in as'$:
      - Replace $v_{a'}$ with $v_{a'} + 1$ in $as'$
    - Add $(f, -, as'', op, c, v'_f)$ to $acc$
  - Invoke $\underline{updateAccessStructureOfResource_{adm}(f, acc)}$

$\underline{assignAccessStructureToResource_{adm}(f, as, op)}$
- If $\exists (f, -, -, op, -, -) \in \mathbf{FA_c}$:
  - $\perp$
- Set $as' = as \vee adm\_att$
- For all $a \in as'$:
  - Find $(a, -, v_a, \mathbf{S2}) \in \mathbf{A_c}$
  - Concatenate $a$ with $v_a$ in $as'$
- Find $(f, \mathbf{p}_{(f, v_f)}, v_f, -, \mathbf{Enc}, \mathbf{S2})$ in $\mathbf{F_c}$ and $(f, \mathbf{p}_{(f, v_f)}, -, op', -, v_f)$ in $\mathbf{FA_c}$
- For all $(f, \mathbf{p}_{(f, v_f)}, -, op', c', v'_f) \in \mathbf{FA_c}$:
  - If $\mathbf{Enc_c} \in \mathbf{Enc}$:
    - Get $\mathbf{MPK}$
    - Decrypt $\mathbf{k}^{\mathbf{sym}}_{(f, v'_f)} = \mathbf{Dec^{ABE}}(\mathbf{MPK}, \mathbf{sk}_{adm}, c')$
    - Set $c'' = \mathbf{Enc^{ABE}}(\mathbf{MPK}, as', \mathbf{k}^{\mathbf{sym}}_{(f, v'_f)})$
  - Else:
    - Set $c'' = \mathbf{N}$
  - Add $(f, \mathbf{p}_{(f, v_f)}, as', op, c'', v'_f)$ to $\mathbf{FA_c}$
- Add $(\langle f, op, \rangle as')$ to $\mathbf{FA_t}$

$\underline{updateAccessStructureOfResource_{adm}(f, acc)}$
- Get $\mathbf{MPK}$
- Find $(f, -, v_f, t_f, \mathbf{Enc}, \mathbf{S2})$ in $\mathbf{F_c}$
- Generate new pseudonym $\mathbf{p}_{(f, v_f+1)} \leftarrow \mathbf{Gen^{Pse}}$
- Replace $(f, -, v_f, t_f, \mathbf{Enc}, \mathbf{S2})$ with $(f, \mathbf{p}_{(f, v_f+1)}, v_f + 1, t_f, \mathbf{Enc}, \mathbf{S2})$ in $\mathbf{F_c}$
- If $\mathbf{Enc_c} \in \mathbf{Enc}$:
  - Generate new symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f, v_f+1)} \leftarrow \mathbf{Gen^{Sym}}$
- If $\exists\, op \in \mathbf{OP}$ *s.t.* $|\{(f, -, -, op, -, -) \in acc\}| = t_f$:
  - If $\mathbf{Enc_c} \in \mathbf{Enc}$:
    - For all $(f, c', v'_f)$ in the DM:
      - Find $(f, -, -, -, c, v'_f) \in \mathbf{FA_c}$
      - Decrypt $\mathbf{k}^{\mathbf{sym}}_{(f, v'_f)} = \mathbf{Dec^{ABE}}(\mathbf{MPK}, \mathbf{sk}_{adm}, c)$
      - Decrypt $fc = \mathbf{Dec}^{\mathbf{S}}_{\mathbf{k}^{\mathbf{sym}}_{(f, v'_f)}}(c')$
      - Replace $(f, c', v'_f)$ with $(f, \mathbf{Enc}^{\mathbf{S}}_{\mathbf{k}^{\mathbf{sym}}_{(f, v_f+1)}}(fc), v_f + 1)$ in the DM
  - Delete all $(f, -, -, -, -, -)$ from $\mathbf{FA_c}$
- Else:
  - For all $(f, -, as, op', c'', v''_f) \in acc$:
    - If $\mathbf{Enc_c} \in \mathbf{Enc}$:
      - Set $c''' = \mathbf{Enc^{ABE}}(\mathbf{MPK}, as, \mathbf{Dec^{ABE}}(\mathbf{MPK}, \mathbf{sk}_{adm}, c''))$
    - Else:
      - Set $c''' = \mathbf{N}$
    - Replace $(f, -, as, op', c'', v''_f)$ with $(f, \mathbf{p}_{(f, v_f+1)}, as, op', c''', v''_f)$ in $\mathbf{FA_c}$
- For all $\{(op, as) | (f, -, as, op, -, -) \in acc\}$:
  - If $\mathbf{Enc_c} \in \mathbf{Enc}$:
    - Set $c'''' = \mathbf{Enc^{ABE}}(\mathbf{MPK}, as, \mathbf{k}^{\mathbf{sym}}_{(f, v_f+1)})$
  - Else:
    - Set $c'''' = \mathbf{N}$
  - Add $(f, \mathbf{p}_{(f, v_f+1)}, as, op, c'''', v_f + 1)$ to $\mathbf{FA_c}$
  - Replace $(f, -, op)f$ with $(\langle f, op \rangle, as)$ in $\mathbf{FA_t}$

$\underline{revokeAccessStructureFromResource_{adm}(f, op)}$
- If $\nexists (f, -, -, op, -, -) \in \mathbf{FA_c}$:
  - $\perp$
- Delete all $(f, -, -, op, -, -)$ from $\mathbf{FA_c}$
- Set $acc = \emptyset$
- For all $(f, -, as, op', -, -) \in \mathbf{FA_c}$:
  - Add $(op', as)$ to $acc$
- Invoke $\underline{updateAccessStructureOfResource_{adm}(f, acc)}$

Figure 4.3: Pseudocode for the administrative operations of the CAC scheme for ABAC

As said in Section 4.2.4, our ABAC CAC scheme allows for specifying which kind of enforcement **Enc** — either cryptographic $\mathbf{Enc_c}$, centralized $\mathbf{Enc_t}$, or both — to apply over a resource. As a result, we can identify two AC policy states, i.e., the one enforced cryptographically $\mathbf{P_c}$ and the one enforced

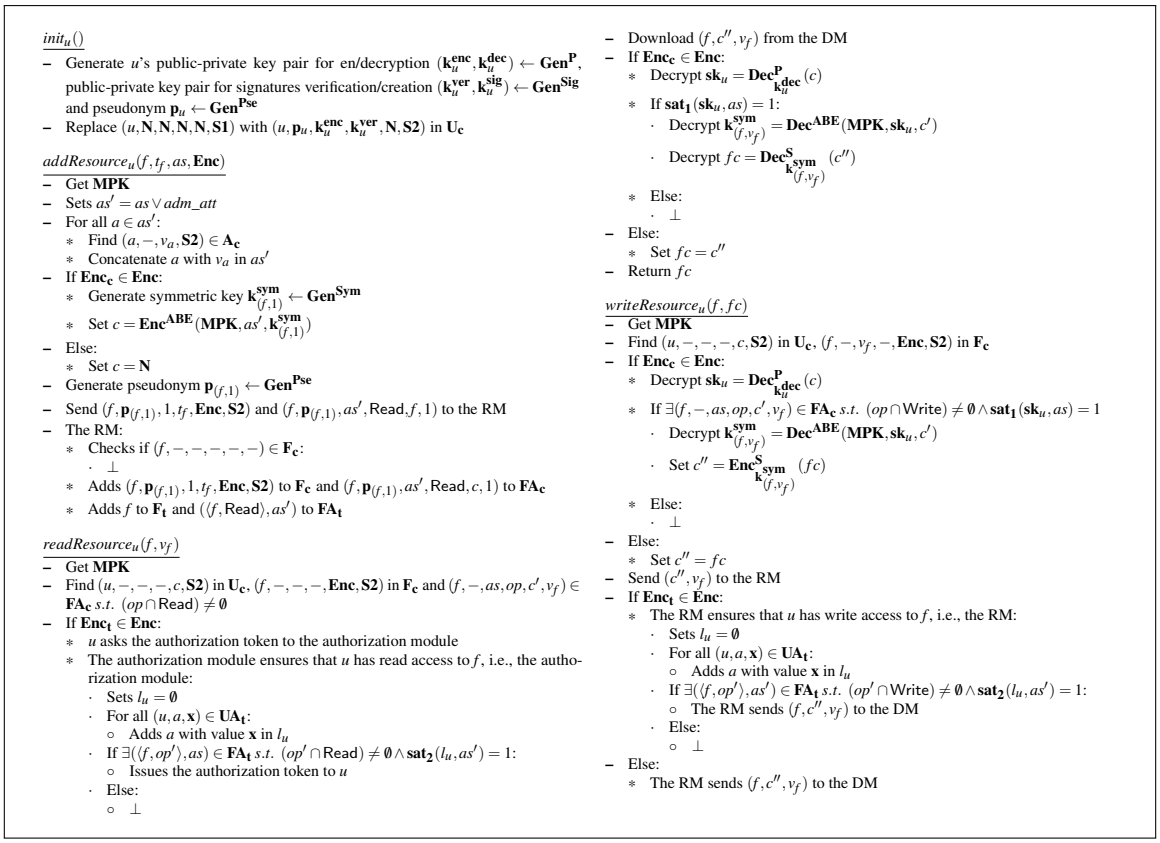


Figure 4.4: Pseudocode for the user operations of the CAC scheme for ABAC

traditionally $\mathbf{P_t}$. The administrator digitally signs both states to provide integrity and authenticity, while the integrity of data is guaranteed assuming the use of AEAD as the type for symmetric cryptosystem [6]; for the sake of simplicity, in Figures 4.3 and 4.4 we omit these details, as well as other trivial checks like the uniqueness of identifiers and pseudonyms. Also, to keep the pseudocode short, we avoid replacing identifiers with pseudonyms in access structures and ABE secret keys. Then, sometimes we use the set intersection symbol ($\cap$) for a set $l$ and an access structure $as$ (i.e., $l \cap as$) to denote the set of attributes present in both $l$ and $as$.

As a final remark, we highlight that in our ABAC CAC scheme, accountability is currently not ensured cryptographically, as already explained in Section 4.3.2. If necessary, users can be required to digitally sign ciphertexts using their private signing keys, at the cost of increasing the computational overhead at the client-side — the modification to the ABAC CAC scheme for implementing this behavior would be straightforward.

**Consistency Between $\mathbf{P_c}$ and $\mathbf{P_t}$.** Similarly to the RBAC CAC scheme (see the end of Section 4.3.2), we use invariants to show that $\mathbf{P_c}$ and $\mathbf{P_t}$ are synchronized on the authorization conditions depending on the ABAC model presented in Section 4.1.2. In detail, any operation in Figures 4.3 and 4.4 leaves the following predicates invariant:

- $(u, -, -, -, -, -) \in \mathbf{U_c} \iff u \in \mathbf{U_t}$;
- $(a, -, -, -) \in \mathbf{A_c} \iff a \in \mathbf{A_t}$;
- $(f, -, -, -, -, -) \in \mathbf{F_c} \iff f \in \mathbf{F_t}$;
- $(u, a, -) \in \mathbf{UA_c} \iff (u, a) \in \mathbf{UA_t}$;

- $(f,-,as,op,-,-) \in \mathbf{FA_c} \iff (\langle f,op\rangle,as) \in \mathbf{FA_t}$.

By using the invariants above and recalling that — given a state $\langle \mathbf{U_t},\mathbf{A_t},\mathbf{F_t},\mathbf{UA_t},\mathbf{FA_t}\rangle$ — a user $u$ can use a permission $\langle f,op\rangle \iff \exists(\langle f,op\rangle,as) \in \mathbf{FA_t}\, s.t.\ as \subseteq \mathcal{P}(\{(a,\mathbf{x})|(u,a,\mathbf{x}) \in \mathbf{UA_t}\})$, it is easy to see that — given a state $\langle \mathbf{U_c},\mathbf{A_c},\mathbf{F_c},\mathbf{UA_c},\mathbf{FA_c}\rangle$ — the user $u$ can use the permission $\langle f,op\rangle \iff \exists(f,-,as,op,\mathbf{Enc}^{\mathbf{ABE}}(\mathbf{MPK},as,\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}),-) \in \mathbf{FA_c}\, s.t.\ as \subseteq \mathcal{P}(\{(a,\mathbf{x})|(u,a,\mathbf{x}) \in \mathbf{UA_c}\})$.

**Example of Execution for the $\underline{deleteUser_{adm}}$ Operation.** Many operations of our ABAC CAC scheme are easily understandable given the discussion in Section 4.2. To illustrate some of the more complex facets of the ABAC CAC scheme, here we describe how to delete a user $u$, an operation which involves (1) the removal of $u$'s information, (2) the revocation of $u$'s attributes and (3) the update of involved resources and the re-keying of resources previously accessible by $u$:

1. we remove $u$ from $\mathbf{U_t}$ and mark $u$'s entry $(u,-,\mathbf{k}^{\mathbf{enc}}_u,\mathbf{k}^{\mathbf{ver}}_u,\mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_u}(\mathbf{sk}_u),\mathbf{S2})$ as "deleted" (**S3**) in $\mathbf{U_c}$; we do not remove the entry as we may need $\mathbf{k}^{\mathbf{ver}}_u$ to verify digital signatures, and also to ensure that usernames are unique (see the $\underline{addUser_{adm}(u)}$ operation in Figure 4.3);

2. we identify the set $l$ of all the attributes assigned to $u$, i.e., $l = \{(a,-)|(u,a,-) \in \mathbf{UA_c}\}$. For each attribute $a$ in $l$, we revoke $a$ from $u$ by deleting the $(u,a,-)$ assignment from both $\mathbf{UA_c}$ and $\mathbf{UA_t}$, increasing $v_a$ by 1 and generating a new pseudonym $\mathbf{p}_{(a,v_a+1)} \Leftarrow \mathbf{Gen}^{\mathbf{Pse}}$. Then, we update $a$'s information — the new pseudonym and version number — in $\mathbf{A_c}$. In this way, $u$'s old secret ABE key $\mathbf{sk}_u$ cannot be used to access new ABE ciphertexts, as these will contain the updated attribute version numbers. Also, we identify the set $us$ of users other than $u$ which are associated with one or more attributes in $l$, i.e., $us = \{u'|(a,-) \in l \wedge (u',a,-) \in \mathbf{UA_c} \wedge u' \neq u\}$, and re-generate their ABE keys to embed the new attributes version numbers;

3. we identify the set $res$ of resources of which one or more access structures contain at least one of the attributes in $l$, i.e., $res = \{f|(f,-,as,-,-,v_f) \in \mathbf{FA_c} \wedge (l \cap as) \neq \emptyset\}$. For each resource $f$ in $res$, we increase $v_f$ by 1 and generate a new pseudonym $\mathbf{p}_{(f,v_f+1)} \Leftarrow \mathbf{Gen}^{\mathbf{Pse}}$. Then, we update $f$'s information — the new pseudonym and version number — in $\mathbf{F_c}$. We also update $f$'s permission-access structures assignments — those whose access structure $as$ contains at least one of the attributes in $l$ — with the new pseudonym $\mathbf{p}_{(f,v_f+1)}$, re-encrypting the symmetric key of the assignment under $as$ updated with the new attribute version numbers. In this way, $u$ cannot use $\mathbf{sk}_u$ to access $f$'s old symmetric keys anymore; however, $u$ could have previously cached them. For this reason, we also generate a new symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f+1)} \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$ which will be used to encrypt new data of $f$, adding the corresponding new assignment to $\mathbf{FA_c}$. Finally, if the number of re-keys for $f$ is equal to the threshold number $t_f$ (recall the discussion in Section 4.2.6), we decrypt and re-encrypt all old content of $f$ with the new key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f+1)}$, removing then old permission-access structure assignment from $\mathbf{FA_c}$.

As a final remark, we highlight that deleted users, as well as deleted resources, cannot be restored. In other words, each user and resource is uniquely identified by a name that cannot be reused. Contrarily, the administrator can restore a previously deleted attribute $a$ (see the $\underline{addAttribute_{adm}(a)}$ action in Figure 4.3), paying attention to generate a new pseudonym and starting counting from the latest version number of $a$ (to avoid old users to use cached ABE keys). However, this is just an arbitrary choice and we note that, if required by the underlying scenario, it is trivial to support both behaviors (i.e., uniqueness of names and restorability) for users, attributes, and resources.

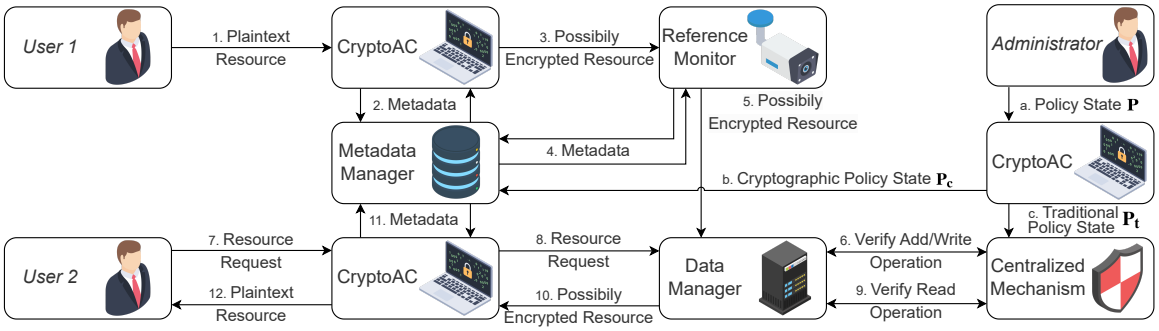


Figure 4.5: Interactions of administrator and users with CryptoAC

## 4.5 CryptoAC

We implement the pseudocode of the CAC schemes for RBAC (see Figures 4.1 and 4.2) and ABAC (see Figures 4.3 and 4.4) into an open-source CAC enforcement mechanism named CryptoAC,[2] using Kotlin multiplatform as the programming language[3] and the React framework for the user interface.[4] CryptoAC is designed as a microservice (e.g., well-defined RESTful APIs — recall the characteristics of microservices in Section 1.1) and is distributed as a Docker[5] image to guarantee seamless integration with cloud native applications: potentially, CryptoAC may also be used as an Software Development Kit (SDK) or plugin for existing software. Indeed, the multiplatform capability of Kotlin allows for Java Virtual Machine (JVM)-based, native, and mobile (e.g., Android and IoS) deployments of CryptoAC. CryptoAC is a stateless microservice, meaning that users can easily export and import their profiles — where, essentially, a profile contains the users' private keys — from and into multiple instances of CryptoAC. As a result, a user may interact with multiple instances of CryptoAC, and the same instance of CryptoAC can serve several users at the same time.

### 4.5.1 Interactions and Flow of Information in CryptoAC

Within the architectural model specified in Section 2.2, CryptoAC acts as the proxy, performing cryptographic computations, interfacing users with resources, and allowing the administrator to manage the AC policy. As shown in Figure 4.5, the *Administrator* provides the AC policy state $\mathbf{P}$ to CryptoAC (step *a*) which then configures the AC policy state enforced cryptographically $\mathbf{P_c}$ (step *b*) and the AC policy state enforced traditionally (step *c*) $\mathbf{P_t}$ accordingly. A user (*User 1* in Figure 4.5) can add or write over a resource by interacting with (the same or a different instance of) CryptoAC (step 1): after retrieving the necessary metadata from the MM (step 2), CryptoAC possibly encrypts the resource — according to the specified enforcement type — and forwards it to the RM along with necessary additional information — see the $\underline{addResource_u}$ and $\underline{writeResource_u}$ operations in Figures 4.2 and 4.4 (step 3). The RM verifies the compliance of the users' AC request (step 4) and forwards the (possibly encrypted) resource to the DM (step 5). Finally, if traditional enforcement is expected for the resource, the DM asks a centralized AC enforcement mechanism to verify the compliance of the users' AC request (step 6). A user (*User 2* in Figure 4.5) can read a resource by interacting with CryptoAC (step 7) which requests the resource to the DM (step 8). According to the enforcement type of the resource, the DM asks the centralized mechanisms to verify the compliance of the users' AC request (step 9) and returns the (possibly encrypted) resources to CryptoAC (step

[2]CryptoAC is an open-source tool for the E2E protection of sensitive data through cryptographic enforcement of access control policies (https://github.com/stfbk/CryptoAC)

[3]Kotlin Multiplatform | Kotlin Documentation (https://kotlinlang.org/docs/multiplatform.html)

[4]React (https://react.dev/)

[5]Docker (https://www.docker.com/)

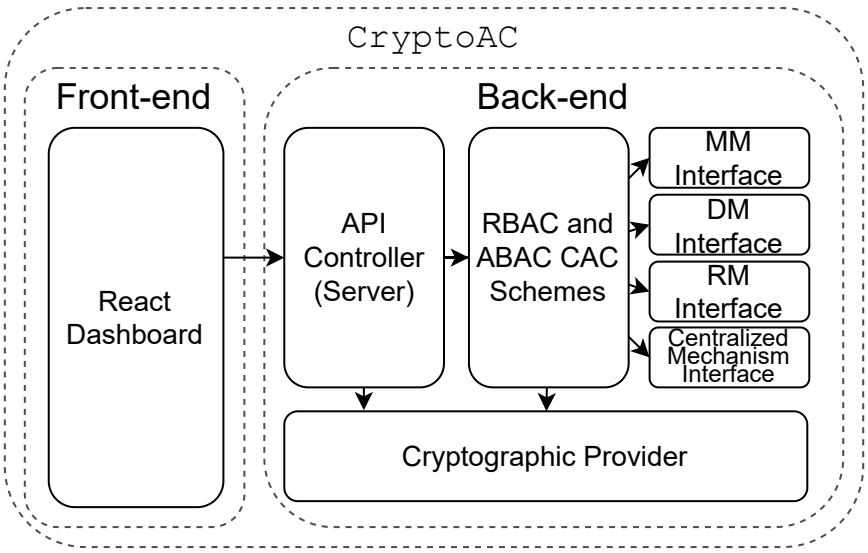


Figure 4.6: Implementation architecture of CryptoAC

10). Finally, CryptoAC retrieves the necessary metadata from the MM (step 11) and returns the requested resource to the user (step 12).

Intuitively, the interactions shown in Figure 4.5 may vary according to the low-level implementation and specific configuration of CryptoAC. For instance, users may send updated resources directly to the DM rather than to the RM to reduce latency and avoid a potential bottleneck (e.g., consider a placement in which the RM and the DM microservices are assigned to two different computing regions). In this case, it is the DM that asks the RM to verify the users' AC request, making the updated resources available only upon a successful verification. Moreover, as explained in Section 2.2.2, the RM may not be present when resources are created by the administrator and cannot be overwritten but only appended by users (e.g., as in [11]). In other words, CryptoAC can be configured for specific scenarios (e.g., considering data at rest, in transit, or both), hence the semantic of users AC requests involving resources (i.e., add, read, and write resource) may vary. For instance, reading a resource when data are at rest may consist in downloading — and possibly decrypting — a file from a simple storage service in the cloud, while reading a resource when data are in transit may consist in subscribing to an MQTT topic. In any case, we highlight that the encoding of the AC policy and the sequence of cryptographic computations done by users to access the symmetric key of a resource (recall the discussion in Sections 4.3.1 and 4.4.1) do not change.

### 4.5.2 Implementation Architecture of CryptoAC

As shown in Figure 4.6, the implementation architecture of CryptoAC is highly modular and split into self-contained components to maximize cohesion and minimize coupling. In other words, to comply with the architectural model, CryptoAC defines high-level interfaces to interact with centralized AC enforcement mechanisms and the other CAC entities, i.e., RM, MM, and DM. CryptoAC exposes its functionalities through RESTful APIs — returning JSON-formatted responses and documented with the OpenAPI specification[6] — that may be invoked either directly over HTTP or through a React user interface. The API Controller is developed with Ktor[7] and interacts directly with a component implementing the RBAC and ABAC CAC schemes. Such a component interacts with the centralized mechanism, the RM, the MM, and the DM through high-level interfaces; this implementation pattern allows supporting several implementations, as shown in Table 4.5. In brief, CryptoAC can interact

[6] OpenAPI Specification - Version 3.0.3 | Swagger (https://swagger.io/specification/)
[7] Ktor (https://ktor.io/)

with both MySQL[8] and Redis[9] as MM; the former is a popular SQL database, the latter is a (primarily in-memory) key-value store. As for the DM, `CryptoAC` can interact with simple storage services for the cloud (e.g., AWS S3[10]) and MQTT brokers — we choose MQTT as it is one of the most widely adopted standards for asynchronous publish/subscribe communication in cloud native applications [88] (recall the discussion on communication patterns in Section 1.1). Since the RM needs to carry out functions specific to CAC, we provide a dedicated implementation. Finally, as centralized AC enforcement mechanisms, `CryptoAC` can interact with OPA, XACML, and also DynSec, which is a centralized AC enforcement mechanism for the Mosquitto MQTT broker.[11] As it is possible to see from Table 4.5, some interface implementations are still missing (i.e., those marked with the symbol "✗"). However, we highlight that this is just an implementation effort, as the main contribution lies in the definition of the interfaces. More details on the interface implementations can be found in the dedicated documentation of `CryptoAC`.[12]

### 4.5.3 Cryptographic Provider for `CryptoAC`

As cryptographic provider, we choose Sodium,[13] a modern and portable cryptographic library whose implementation was thoroughly audited, and no flaws or vulnerabilities were found.[14] Sodium uses the Elliptic-Curve Diffie-Hellman (ECDH) algorithm (X25519) to generate public-private key pairs and the Edwards-curve Digital Signature Algorithm (EdDSA) for digital signatures (Ed25519). Like many ciphers (e.g., those in TLS 1.3), Sodium supports AEAD with the XSalsa20 symmetric stream cipher (i.e., Salsa20 with 192-bit nonce extension) together with the Poly1305 universal hash function instead of 256-bit Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) with, e.g., the SHA-384 hash function. One of the reasons for this choice is that, although hardware acceleration for AES is often available in modern processors, its performance on platforms that lack such hardware is considerably lower. Another issue is that many software-only AES implementations may be vulnerable to cache-collision timing attacks [13]. Instead, XSalsa20 is faster than (non-accelerated) AES and provides homogeneous performance across similar hardware, enhancing portability.

[8] MySQL (https://www.mysql.com/it/)
[9] Redis (https://redis.io/)
[10] Cloud Object Storage - Amazon S3 (https://aws.amazon.com/s3/)
[11] Dynamic Security Plugin | Eclipse Mosquitto (https://mosquitto.org/documentation/dynamic-security/)
[12] CryptoAC Documentation (https://cryptoac.readthedocs.io/)
[13] Sodium (https://libsodium.gitbook.io/doc)
[14] Libsodium Audit Results (https://www.privateinternetaccess.com/blog/libsodium-audit-results/)

Table 4.5: Currently supported interface implementations in `CryptoAC`

| Entity | Implementation | CAC scheme | |
|---|---|---|---|
| | | RBAC | ABAC |
| MM | MySQL | ✓ | ✓ |
| | Redis | ✓ | ✗ |
| DM | Simple storage services | ✓ | ✓ |
| | MQTT brokers | ✓ | ✓ |
| RM | Dedicated component | ✓ | ✓ |
| Centralized Mechanism | OPA | ✓ | ✗ |
| | XACML | ✓ | ✓ |
| | DynSec | ✓ | ✗ |

As the cryptographic provider for ABE, we choose the OpenABE C/C++ software library[15] which — for CP-ABE — uses a variant of the Waters Large Universe system [126] instantiated with the (pairing friendly) elliptic curve BN-254. As the last update for OpenABE at the time of writing was in 2023, we fork its GitHub repository, fixing the installation scripts and updating its dependencies on other software libraries (i.e., Relic,[16] and OpenSSL[17]. Then, we develop bindings for Kotlin allowing to easily use OpenABE from CryptoAC (and any Kotlin program). Besides CryptoAC, we made the updated version of OpenABE[18] and the bindings for Kotlin[19] available online as open-source software, so that other researchers and practitioners can more easily experiment with ABE.

### 4.5.4 Security Comparison with Transport Layer Security

We now briefly compare the security properties offered by CryptoAC with those offered by one of the most widely adopted security mechanisms to secure communications in cloud native applications, i.e., TLS (see Table 4.6). Below, we define each security property and then explain whether CryptoAC and TLS provide that property.

**Confidentiality.** The property for which a given resource can be accessed by authorized users only. In this regard, CryptoAC provides E2E confidentiality while TLS offers hop-to-hop confidentiality only, hence not preventing unauthorized access by participating parties (e.g., honest-but-curious cloud providers storing data at rest or operating MQTT brokers mediating communications).

**Integrity.** The property for which any alteration of a resource is immediately noticeable. Both CryptoAC and TLS provide integrity by using MACs in AEAD (see Section 4.2.3 and [6]). However, while CryptoAC provides E2E integrity, TLS offers hop-to-hop integrity only. In other words, when using TLS, participating parties mediating communications could potentially modify resources without users noticing.

**Network-Level Security Properties.** We consider two security properties at the network level: forward secrecy and replay attack protection. The former is the property for which the compromise of a user's private key — where the compromise happens in the future — does not compromise the confidentiality of network messages (i.e., the payloads) exchanged by that user in the past, while the latter is the property for which — at network level — past network messages cannot be replayed

[15] The OpenABE library - open source cryptographic library with attribute-based encryption implementations in C/C++ (https://github.com/zeutro/openabe)
[16] GitHub - relic-toolkit/relic: Code (https://github.com/relic-toolkit/relic)
[17] OpenSSL (https://www.openssl.org/)
[18] Stefano Berlato - OpenABE (https://github.com/StefanoBerlato/openabe)
[19] Stefano Berlato - Kotlin Multiplatform OpenABE (https://github.com/StefanoBerlato/kotlin-multiplatform-openabe)

Table 4.6: Comparison of security properties offered by CryptoAC and TLS

| Security Property | | CryptoAC | TLS |
|---|---|---|---|
| Confidentiality | Point-to-Point | ✓ | ✓ |
| | E2E | ✓ | ✗ |
| Integrity | Point-to-Point | ✓ | ✓ |
| | E2E | ✓ | ✗ |
| Network-Level | Forward Secrecy | ✗ | ✓ |
| | Replay Attack Protection | ✓ | ✓ |
| Application-Level | Replay Attack Protection | ✓ | n/a |
| | Authorization | ✓ | n/a |

by an attacker in such a way to be accepted by (honest) users. While the new version of TLS (v1.3) provides forward secrecy by default thanks to single-use ephemeral key pairs in the Diffie-Hellman key agreement protocol (see [98]), `CryptoAC` does not provide forward secrecy. Indeed, the compromise of a user $u$'s private key $\mathbf{k}_u^{\mathbf{dec}}$ would allow an attacker to decrypt past (symmetric keys and then) network messages to which $u$ had access at the time. Nonetheless, we note that the impact of an attacker — intended as the number of network messages the attacker can read after having compromised a user's private key — can be reduced by periodically rotating the private keys of users, roles, and resources. Regarding replay attack protection (at the network level), both `CryptoAC` and TLS provide this property thanks to the use of AEAD.

**Application-Level Security Properties.** We consider two security properties at the application level: authorization and replay attack protection. The former is the property for which it is possible to define and enforce AC policies, while the latter is the property for which — at the application level — past network messages cannot be replayed by an attacker in such a way to be accepted by (honest) users. Intuitively, CAC intrinsically provides authorization, whereas TLS does not. Regarding replay attack protection at the application level, our CAC scheme provides this property thanks to the use of AEAD. However, similarly to the integrity security property, we note that, when using TLS, mediating parties could potentially replay network messages without the users noticing. Being TLS a security mechanism at the network level, we mark these two properties as not applicable (n/a) to TLS in Table 4.6.

From Table 4.6 it is easy to see that `CryptoAC`, although providing several properties, is not a "silver bullet" security mechanism. Instead, its adoption may be evaluated on a case-by-case basis according to the underlying scenario (e.g., if forward secrecy is a stringent requirement, then `CryptoAC` — in its current version — may not be an adequate solution standalone ) More importantly, we remark that `CryptoAC` can be used in synergy with other security mechanisms, so to achieve a more complete and throughout protection of sensitive data in cloud native applications.

## 4.6 Experimental Evaluation for Role-Based Cryptographic Access Control

We now present a thorough performance evaluation of the RBAC CAC scheme presented in Section 4.3 and implemented in `CryptoAC` as described in Section 4.5). To this end, we design and conduct four different experiments (*EXP*):

- *EXP-1* - one of the most widely adopted security mechanisms to protect data in transit in cloud native applications is TLS, which expects devices to perform (often cumbersome) handshakes and key derivation algorithms [108, 50, 113]; however, also retrieving symmetric keys of resources in CAC requires noticeable computational effort. Therefore, in this experiment we compare how long it takes for a microservice (e.g., a container running on a laptop or IoT device) to establish a connection with another microservice (e.g., a server such as an MQTT broker) when using CAC and when using TLS with respect to a baseline configuration — i.e., with no security mechanism (Section 4.6.1);

- *EXP-2* - after having established a connection, microservices communicate with each other according to the behavior of the underlying cloud native application. In this context, this experiment concerns the overhead imposed over the communication by the chosen security mechanism — i.e., either TLS or CAC — with respect to the baseline configuration. As a concrete scenario, we consider an IoT-based cloud native application where we measure the

transmission time (i.e., from publisher to subscriber) of a specific number of MQTT messages while varying the number of publishers and subscribers to evaluate the scalability of each security mechanism (Section 4.6.2);

- *EXP-3* - during the lifetime of cloud native applications, AC policies are likely to be dynamically updated; for instance, roles may be added and permissions may be removed. Therefore, in this experiment we measure the performance of administrative operations — that is, those reported in Table 3.2 for which the pseudocode was provided in Figure 4.1 (Section 4.6.3);
- *EXP-4* - as explained in Sections 4.2.4 and 4.3.2, our RBAC CAC scheme considers two policy states, i.e., $\mathbf{P_t}$ and $\mathbf{P_c}$. Therefore, in this experiment we measure how the number of users, roles, resources, user-role, and role-permission assignments affect the size of the policy states (Section 4.6.4).

We make the replication package — that is, the software and the instructions necessary to replicate the experiments — and the results of all experiments available online.[20]

### 4.6.1 *EXP-1* - Communication Setup

We want to measure the overhead on the communication setup between a pair of microservices in three configurations, i.e., when using no security mechanism (i.e., baseline), when using CAC (that is, when asking `CryptoAC` to retrieve the symmetric key of a resource), and when using unilateral TLSv1.3 with a self-signed certificate and the *TLS_CHACHA20_POLY1305_SHA256* cipher (the same cipher of the Sodium cryptographic library used in `CryptoAC` — see Section 4.5.3). The two microservices are an MQTT client and an MQTT broker which we implement, respectively, with `CryptoAC` and the Mosquitto MQTT broker. In addition to the CAC configuration, we use the codebase of `CryptoAC` to implement also the TLS and baseline configurations to avoid introducing biases due to the use of different software across configurations. In detail — and only for this experiment — we remove the cryptographic computations of the RBAC CAC scheme from `CryptoAC`, enabling TLS in the MQTT broker to implement the TLS configuration, and disabling TLS for the baseline configuration. In this way, the infrastructure remains the same across the three configurations, allowing us to more precisely measure the overhead of the chosen cryptographic mechanisms (i.e., either `CryptoAC` or TLS). Moreover, to exclude network latency — which may sensibly vary depending on the scenario — we deploy the microservices as Docker containers on the same device, that is a laptop running Ubuntu 18.04 on an Intel(R) Core(TM) i7-11370H and 16GB of RAM. The experiment thus consists in asking `CryptoAC` — as MQTT client — to connect to the Mosquitto MQTT broker in the three aforementioned configurations; we repeat the experiment 1,000 times for each configuration.

**Results.** We report the obtained results as three box plots in Figure 4.7; each box plot is bounded by lower and upper quartiles, while we compute upper and lower whiskers as the default — i.e., 1.5× upper and lower interquartile range. We represent the data points for each configuration as grey dots, omitting those points outside of the whiskers range (i.e., the outliers). The red dot in each box plot represents the median value for each configuration: 402.204ms for `CryptoAC`, 400.998ms for TLS and 400.993ms for the baseline. This means that, while TLS adds a negligible overhead to the communication setup, `CryptoAC` adds an overhead of 1.211ms. We believe that the better performance of TLS may be (partly) because the TLS session is unilateral and the certificate of the broker is self-signed, hence the MQTT client is not required to verify the entire chain of trust (e.g., from intermediate authorities' certificates to the root certificate) to establish the authenticity and

[20] replication package Chapter 4 (https://drive.google.com/file/d/1A2skgu9E96P4sA25_QnK-EEr6ZhkaXii/view?usp=sharing)

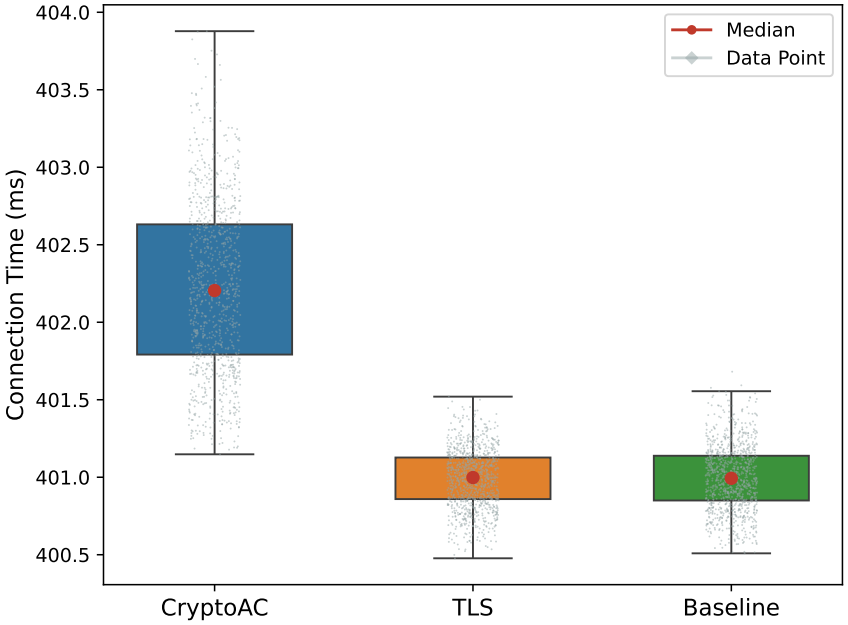


Figure 4.7: Overhead of the communication setup for `CryptoAC`, TLS, and the baseline configurations

validity of the certificate. Regardless, we believe that the overhead of `CryptoAC` (i.e., 1.211ms) is still acceptable in the majority of IoT-based cloud native applications, especially when considering that this overhead incurs just once, as `CryptoAC` caches the symmetric keys of resources at the client-side. Intuitively, the cache is invalidated when symmetric keys are renewed (e.g., after a revocation). Finally, we believe that this overhead can be reduced by optimizing the implementation of `CryptoAC` and fine-tuning the parameters of the cryptographic algorithms employed; we leave the verification of these ideas as future work (see Section 5.1.3).

### 4.6.2 *EXP-2* - Communication Overhead

We want to evaluate the scalability of the three configurations — namely CAC, TLS (v1.3, unilateral, with a self-signed certificate and using the *TLS_CHACHA20_POLY1305_SHA256* cipher), and the baseline configuration — using the same experimental settings of *EXP-1* (see Section 4.6.1). To this end, we measure the transmission time of a specific number of MQTT messages and, to evaluate scalability, we repeat the measurement while varying the number of publishers and subscribers (see Table 4.7). We define the transmission time of an MQTT message as the time elapsed from the publishing of the message in a topic and the receival of the message by a subscriber of that topic — intuitively, the transmission time includes cryptographic computations, e.g., of CAC or TLS. Consequently, the scalability of a configuration can be evaluated by considering the (relative) increment in the transmission time among repeated measurements, i.e., when increasing the number of publishers and subscribers. In this regard, we expect CAC to exhibit higher scalability with respect to TLS, the reason being that, with TLS, the MQTT broker de/encrypts each MQTT message for

Table 4.7: Median Transmission Times (in ms) in *EXP-2*

| Configuration | Number of Publishers (note that there are as many subscribers) | | | | | |
|---|---|---|---|---|---|---|
| | 5 | 10 | 15 | 20 | 25 | 30 |
| CryptoAC | 2.0 | 2.0 | 2.0 | 2.0 | 3.0 | 3.0 |
| Baseline | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 2.0 |
| TLS | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 3.0 |

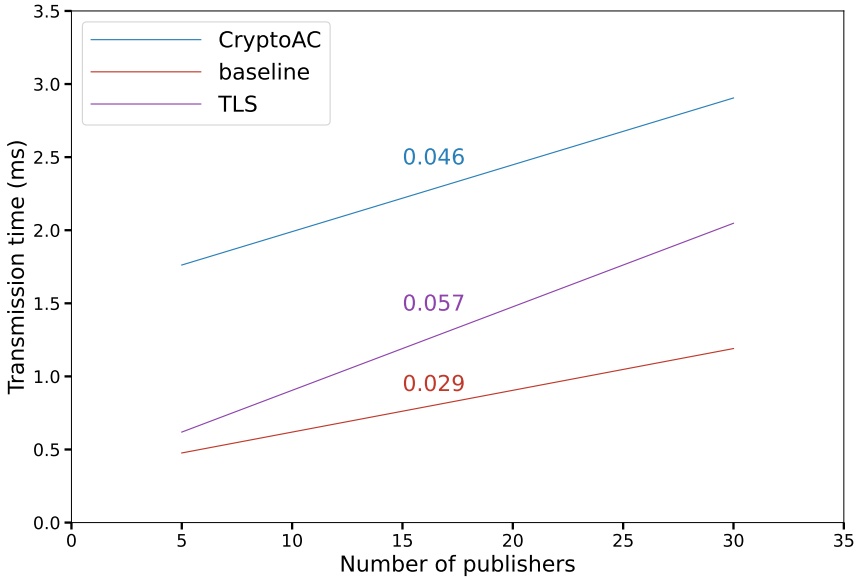


Figure 4.8: Overhead of the communication for `CryptoAC`, TLS, and the baseline configurations

each publisher/subscriber individually using the corresponding symmetric session key, while in CAC cryptographic computations are distributed among MQTT clients (i.e., users).

As in *EXP-1*, to exclude network latency, we deploy multiple instances of `CryptoAC` (i.e., the publishers and the subscribers) and the Mosquitto MQTT broker on the same device, that is a *c6i.16xlarge* virtual machine (64 vCPUs, 128 GB RAM and 8GB SSD) running Ubuntu Server 22.04 LTS provisioned through the AWS EC2 service;[21] we choose this device to ensure that each Docker container can run in a dedicated CPU to avoid competition over computational resources. Notably, to prevent the Mosquitto MQTT broker from discarding MQTT messages, we configure it to support an unlimited number of messages in the process of being transmitted (instead of 20, the default threshold) and queued for transmission (instead of 1000, the default threshold) via the *max_inflight_messages* and *max_queued_messages* parameters, respectively.[22] If not, messages exceeding the default thresholds would otherwise be discarded by the broker — regardless of their quality of service value[23] — potentially leading to biased results. Notably, to measure transmission time, we use `ACME` (see Section 3.4); we refer the interested reader to the replication package for more details on the experimental setup.

The experiment thus consists of asking half the instances of `CryptoAC` — as subscribers — to subscribe to a topic, and the other half of the instances of `CryptoAC` — as publishers — to publish each 500 messages (with quality of service 1) to that topic; we repeat the experiment in each of the three aforementioned configurations scaling the number of publishers and subscribers from 5 to 30, in steps of 5 (the upper limit accounts for the number of vCPUs available in the virtual machine). For instance, consider the run with 5 publishers and 5 subscribers (i.e., a total number of 10 instances of `CryptoAC`): after establishing a connection to the broker (which we do not measure), the 5 subscribers subscribe to a topic with quality of service 1, and then the 5 publishers each send 500 MQTT messages with quality of service 1 to that topic; the total number of MQTT messages amounts then at 12,500 (i.e., 5 publishers sending 500 messages to 5 subscribers each, that is, $5 \cdot 500 \cdot 5$).

**Results.** We report the obtained results in Figure 4.8, together with the linear regressions (blue, purple, and red solid lines for `CryptoAC`, TLS, and baseline, respectively; slopes — or gradients —

[21] Amazon EC2 C6i Instances - Amazon Web Services (https://aws.amazon.com/ec2/instance-types/c6i/)
[22] mosquitto.conf man page (https://mosquitto.org/man/mosquitto-conf-5.html)
[23] MQTT man page | Eclipse Mosquitto (https://mosquitto.org/man/mqtt-7.html)

0.046, 0.057, and 0.029, respectively) computed on the median transmission times for each run of the three configurations — we report the median transmission times in Table 4.7. The results show that, in the baseline configuration, the median transmission time of MQTT messages increases by 0.029ms each time 5 publishers and 5 subscribers are added, achieving the best results in terms of both median transmission time and scalability. Then, from Figure 4.8, we infer that `CryptoAC` generally performs worse than TLS, as indicated by the fact that the median transmission times for `CryptoAC` are always (equal or) higher than those of TLS. Still, the difference in performance is limited to at most 1-2ms, an overhead acceptable in the majority of IoT applications and often dominated by other factors (e.g., network latency). Finally, as reported in Section 4.6.1, the implementation of `CryptoAC` can be further optimized. However, the most notable result is that `CryptoAC` does exhibit higher scalability than TLS (i.e., $0.046 < 0.057$), as theorized at the beginning of Section 4.6.2. As a final remark, we note that the regression lines indicate that `CryptoAC` would theoretically perform better than TLS at 110 publishers (i.e., $0.046 \cdot x + 1.533 < 0.057 \cdot x + 0.333$ when $x = 110$); we leave the verification of this hypothesis — which would require dedicated hardware with non-negligible costs — as future work (see Section 5.1.3).

### 4.6.3 *EXP-3* - Administrative Operations

We evaluate the performance of the administrative operations in our RBAC CAC scheme from three points of view: *algebraic cost*, *cryptographic time* and *implementation time*; we then report the obtained results in Table 4.8.

**Algebraic Cost.** This is the number and type of cryptographic computations involved in each administrative operation. We extract the algebraic costs from Figure 4.1, considering digital signature computations as well — i.e., a create signature computation $\mathbf{Sign^P}$ for each element added to $\mathbf{P_c}$ and a verify signature computation $\mathbf{Ver^P}$ for each element retrieved from $\mathbf{P_c}$. For simplicity, we represent the cost of a cryptographic computation with the symbol of the computation itself (e.g., $\mathbf{Gen^P}$ represents the cost of generating an asymmetric encryption key pair). Then, we use the symbol $|\cdot|$ to indicate the cardinality of a set (i.e., the number of its elements).

Table 4.8: Performance evaluation of administrative operations (in milliseconds)

| Operation | Algebraic Cost | Cryptographic Time | Implementation Time |
|---|---|---|---|
| $\underline{init_{adm}()}$ | $\mathbf{Gen^P} + \mathbf{Gen^{Sig}} + \mathbf{Enc^P} + (3 \cdot \mathbf{Sign^P})$ | 0.301ms | 0.773ms |
| $\underline{addUser_{adm}(u)}$ | $\mathbf{Sign^P}$ | 0.047ms | 0.759ms |
| $\underline{deleteUser_{adm}(u)}$ | $(\mathbf{Ver^P} + \underline{revokeUserFromRole_{adm}(u,r)}) \cdot \lvert\{r : (u,r,-,-) \in \mathbf{UR_c}\}\rvert$ | | |
| $\underline{addRole_{adm}(r)}$ | $\mathbf{Gen^P} + \mathbf{Gen^{Sig}} + \mathbf{Enc^P} + (2 \cdot \mathbf{Sign^P})$ | 0.254ms | 1.359ms |
| $\underline{deleteRole_{adm}(r)}$ | $(\mathbf{Ver^P} + \underline{revokePermissionFromRole_{adm}(r,\langle f,op\rangle)}) \cdot \lvert\{f : (r,f,-,-,-,-,-,-) \in \mathbf{PA_c}\}\rvert$ | | |
| $\underline{addResource_{adm}(f,fc,\mathbf{Enc_c})}$ | $(2 \cdot \mathbf{Sign^P}) + \mathbf{Gen^{Sym}} + \mathbf{Enc^P} + \mathbf{Enc^S}$ | 0.195ms + 0.156ms/KB | 1.592ms + 0.156ms/KB |
| $\underline{deleteResource_{adm}(f)}$ | No cryptographic computation involved | | 1.622ms |
| $\underline{assignU_{adm}(u,r)}$ | $\mathbf{Dec^P} + \mathbf{Enc^P} + \mathbf{Sign^P} + \mathbf{Ver^P}$ | 0.305ms | 4.391ms |
| $\underline{assignPermissionToRole_{adm}(r,\langle f,op\rangle)}$ | If $(r,f,-,-,-,-,-,-,-) \notin \mathbf{PA_c}$, then $((3 \cdot \mathbf{Ver^P}) + \mathbf{Sign^P} + \mathbf{Dec^P} + \mathbf{Enc^P})$, else $(\mathbf{Ver^P} + \mathbf{Sign^P})$ | If $(r,f,-,-,-,-,-,-,-) \notin \mathbf{PA_c}$, then 0.485ms, else 0.137ms | If $(r,f,-,-,-,-,-,-,-) \notin \mathbf{PA_c}$, then 5.312ms, else 1.587ms |
| $\underline{revokeUserFromRole_{adm}(u,r)}$ | $\mathbf{Gen^P} + \mathbf{Gen^{Sig}} + \mathbf{Dec^P} + (2 \cdot \mathbf{Ver^P}) + \mathbf{Sign^P} + (\mathbf{Ver^P} + \mathbf{Sign^P} + \mathbf{Enc^P}) \cdot \lvert\{u' : (u',r,-,-) \in \mathbf{UR_c} \wedge u' \neq u\}\rvert + \lvert\{f : (r,f,-,-,-,-,-,-,-) \in \mathbf{PA_c}\}\rvert \cdot (2 \cdot (\mathbf{Ver^P} + \mathbf{Sign^P}) + \mathbf{Gen^{Sym}} + \mathbf{Enc^P} + (2 \cdot \mathbf{Ver^P} + \mathbf{Sign^P} + \mathbf{Enc^P}) \cdot (\lvert\{(r',f') : (r',f',-,-,-,-,-,-,-) \in \mathbf{PA_c} \wedge r' \neq r\}\rvert))$ | 0.359ms + (0.235ms) $\cdot \lvert\{u' : (u',r,-,-) \in \mathbf{UR_c} \wedge u' \neq u\}\rvert$ + $\lvert\{f : (r,f,-,-,-,-,-,-,-) \in \mathbf{PA_c}\}\rvert$ $\cdot$(0.375ms + (0.325ms) $\cdot(\lvert\{(r',f') : (r',f',-,-,-,-,-,-,-) \in \mathbf{PA_c} \wedge r' \neq r\}\rvert)$) | 0.842ms + (1.292ms) $\cdot \lvert\{u' : (u',r,-,-) \in \mathbf{UR_c} \wedge u' \neq u\}\rvert$ + $\lvert\{f : (r,f,-,-,-,-,-,-,-) \in \mathbf{PA_c}\}\rvert$ $\cdot$(3.217ms + (1.649ms) $\cdot(\lvert\{(r',f') : (r',f',-,-,-,-,-,-,-) \in \mathbf{PA_c} \wedge r' \neq r\}\rvert)$) |
| $\underline{revokePermissionFromRole_{adm}(r,\langle f,op\rangle)}$ | $\mathbf{Ver^P}$ + (if $(r,f,-,-,-,-,-,-,op'') \in \mathbf{PA_c} \wedge op'' \subseteq op$, then $(\mathbf{Ver^P} + \mathbf{Gen^{Sym}} + \mathbf{Sign^P} + (2 \cdot \mathbf{Ver^P} + \mathbf{Sign^P} + \mathbf{Enc^P}) \cdot \lvert\{r' : (r',f,-,-,-,-,-,-,-) \in \mathbf{PA_c} \wedge r' \neq r\}\rvert$), else if $op \cap op'' \neq \emptyset$ then $\mathbf{Sign^P}$) | 0.090ms + (if $(r,f,-,-,-,-,-,-,op'') \in \mathbf{PA_c} \wedge op'' \subseteq op$, then (0.140ms + (0.325ms) $\cdot \lvert\{r' : (r',f,-,-,-,-,-,-,-) \in \mathbf{PA_c} \wedge r' \neq r\}\rvert$), else if $op \cap op'' \neq \emptyset$ then 0.047ms) | 0.950ms + (if $(r,f,-,-,-,-,-,-,op'') \in \mathbf{PA_c} \wedge op'' \subseteq op$, then (1.480ms + (3.500ms) $\cdot \lvert\{r' : (r',f,-,-,-,-,-,-,-) \in \mathbf{PA_c} \wedge r' \neq r\}\rvert$), else if $op \cap op'' \neq \emptyset$ then 0.269ms) |

**Cryptographic Time.** This is the execution time obtained by summing the execution time — when run in `CryptoAC` — of each cryptographic computation involved in each administrative operation. As said in Section 4.5.3, we choose the Sodium cryptographic library that uses ECDH-X25519 for asymmetric en/decryption, EdDSA-Ed25519 for asymmetric signature creation/verification and XSalsa20-Poly1305 for symmetric en/decryption. The most data asymmetrically encrypted and decrypted in one computation — thus, the worst case for the performance of $\mathbf{Enc^P}$ and $\mathbf{Dec^P}$ (as shown in Section 4.6.4) — are the four keys of a role in $\mathbf{UR_c}$ (that amount at 160 bytes), while the execution times for $\mathbf{Enc^S}$ and $\mathbf{Dec^S}$ are calculated per KB. Finally, the most data from which signatures (that are 64 bytes long) are generated — thus, the worst case for the performance of $\mathbf{Sign^P}$ and $\mathbf{Ver^P}$ — are 460 bytes (i.e., the biggest block of data that is produced in $\mathbf{P_c}$, as later discussed in Section 4.6.4 and shown in Table 4.10). We use the same experimental settings as in *EXP-1* (see Section 4.6.1) to measure the execution time of cryptographic computations, which we report in Table 4.9; we repeat these measurements 1,000 times each to reduce measuring errors.

**Implementation Time.** This is the execution time of each administrative operation when running `CryptoAC`, including both cryptographic computations, the overhead of the implementation (i.e., accessory code such as consistency checks) and the interactions among entities as well (e.g., connection toward the MM). Again, we use the same experimental settings as in *EXP-1*, and employ `kotlinx.benchmark`[24] as a library for running benchmarks, where each benchmark consists in executing an administrative operation — each benchmark is repeated 1,000 times to reduce measuring errors.

**Results.** We report the obtained results in Table 4.8. From the table, we can see that the implementation time of all administrative operations is reasonably bounded to a few milliseconds (from 0.759ms for $\underline{addUser_{adm}(u)}$ to 5.312ms for $\underline{assignPermissionToRole_{adm}(r,\langle f,op\rangle)}$). However, we can also see that the performance of 4 administrative operations depends on the state of $\mathbf{P_c}$ (e.g., on the number of role-permission assignments). We now explore this aspect more in detail, focusing on $\underline{revokeUserFromRole_{adm}(u,r)}$ and $\underline{revokePermissionFromRole_{adm}(r,\langle f,op\rangle)}$, since the performance of $\underline{deleteUser_{adm}(u)}$ and $\underline{deleteRole_{adm}(r)}$ can be then derived straightforwardly.

**Revoking a User From a Role.** The performance of revoking a user $u$ from a role $r$ — which requires renewing $r$'s keys as well as the keys of all resources $u$ could access through $r$ — is influenced by three parameters: (*i*) the number of other users to which $r$'s new keys have to be distributed (i.e., $|\{u' : (u',r,-,-) \in \mathbf{UR_c} \wedge u' \neq u\}|$), (*ii*) the number of resources for which a new symmetric key has to be distributed (i.e., $|\{f : (r,f,-,-,-,-,-,-,-) \in \mathbf{PA_c}\}|$) and (*iii*) the number of other roles to which the new symmetric keys have to be distributed (i.e., $|\{(r',f') : (r',f',-,-,-,-,-,-,-) \in \mathbf{PA_c} \wedge r' \neq r\}|$). Hence, we measure the performance of the operation $\underline{revokeUserFromRole_{adm}(u,r)}$ by varying these three influencing parameters — one at a time — from 1 to 500 by a step of 5. We

[24] GitHub - Kotlin/kotlinx-benchmark: Kotlin multiplatform benchmarking toolkit (https://github.com/Kotlin/kotlinx-benchmark)

Table 4.9: Execution time of cryptographic computations in `CryptoAC`

| Computation | Time | Computation | Time |
|---|---|---|---|
| $\mathbf{Gen^{Sig}}$ (EdDSA-Ed25519) | 0.031ms | $\mathbf{Gen^P}$ (ECDH-X25519) | 0.031ms |
| $\mathbf{Gen^{Sym}}$ (XSalsa20-Poly1305) | 0.003ms | $\mathbf{Enc^S}$ (XSalsa20-Poly1305) | 0.156ms/KB |
| $\mathbf{Dec^S}$ (XSalsa20-Poly1305) | 0.176ms/KB | $\mathbf{Enc^P}$ (ECDH-X25519) | 0.098ms |
| $\mathbf{Dec^P}$ (ECDH-X25519) | 0.070ms | $\mathbf{Ver^P}$ (EdDSA-Ed25519) | 0.090ms |
| $\mathbf{Sign^P}$ (EdDSA-Ed25519) | 0.047ms | | |

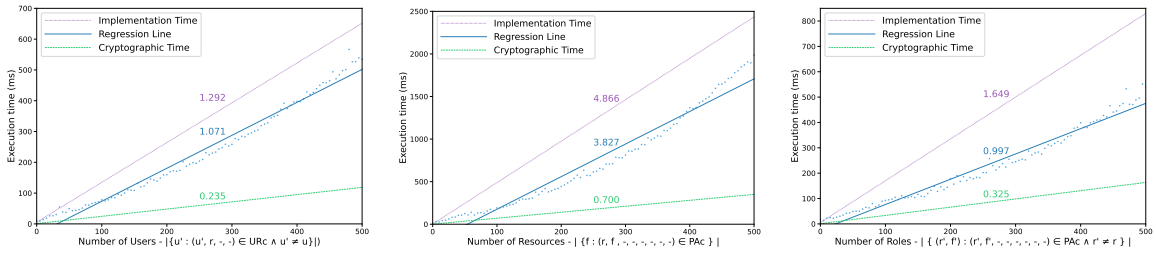


Figure 4.9: Performance of $\underline{revokeUserFromRole_{adm}(u,r)}$

report the results in Section 4.6.3 together with the linear regressions (blue solid lines, slopes — or gradients — 1.071, 3.827 and 0.997, respectively). Moreover, we also report the linear regressions on the "Cryptographic Time" (green dashed lines, slopes 0.235, 0.700 and 0.325, respectively) and the "Implementation Time" (purple dotted lines, slopes 1.292, 4.886 and 1.649, respectively) that we compute by simply multiplying the corresponding values in Table 4.8 for each point on the X axes (i.e., from 0 to 500 by a step of 5).

**Revoking a Permission From a Role.** The performance of revoking all permissions from a role $r$ over a resource $f$ — which requires renewing $f$'s symmetric key — is influenced by 1 parameter, that is the number of other roles to which $f$'s new key has to be distributed, i.e., $|\{r' : (r', f, -, -, -, -, -, -, -) \in \mathbf{PA_c} \wedge r' \neq r\}|$. Hence, we measure the performance of $\underline{revokePermissionFromRole_{adm}(r, \langle f, op \rangle)}$ by varying this influencing parameter from 0 to 500 by a step of 5. We report the results in Figure 4.10 together with the linear regression (blue solid line, slope 1.642). Moreover, we also report the linear regression on the "Cryptographic Time" (green dashed line, slope 3.500) and the "Implementation Time" (purple dotted line, slope 0.325) that we compute by simply multiplying the corresponding values in Table 4.8 for each point on the X axis (i.e., from 0 to 500 by a step of 5).

**Discussion.** From Figures 4.9 and 4.10, we can see that the cryptographic and implementation times are, respectively, a lower and an upper bound to the actual execution times. These results are reasonable, as administrative operations in `CryptoAC` have — in addition to cryptographic computations — computational costs (e.g., connection toward the MM) that, however, are incurred only once, regardless of the state of the policy. Also, the slopes of the linear regressions give an idea of how much costs increase depending on the influencing parameters. For instance, the cost for $\underline{revokePermissionFromRole_{adm}(r, \langle f, op \rangle)}$ increases by 0.979ms (see Figure 4.10) for each role to which $f$'s new key has to be distributed. Overall, the slopes suggest good scalability, especially if considering that the influencing parameters of $\underline{revokeUserFromRole_{adm}(u,r)}$ and $\underline{revokePermissionFromRole_{adm}(r, \langle f, op \rangle)}$ concern the number of users, roles and resources involved in these operations, and not the total number of users, roles and resources in $\mathbf{P_c}$. Finally, we highlight that administrative operations can be scheduled for when the cloud native application is usually idling or unloaded (e.g., during the night).

### 4.6.4 *EXP-4* - Policy States Size

The size of $\mathbf{P_t}$ and $\mathbf{P_c}$ in our RBAC CAC scheme can be easily determined from their encoding (see Sections 4.1.1 and 4.3.1, respectively) and the implementation details of `CryptoAC` such as the cryptographic algorithms employed, thus without running an actual experiment. We manually measure the size of other elements of $\mathbf{P_c}$ — e.g., $\mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_u}\left(\mathbf{k}^{\mathbf{enc}}(r,v_r)\mathbf{k}^{\mathbf{dec}}(r,v_r), \mathbf{k}^{\mathbf{ver}}_{(r,v_r)}, \mathbf{k}^{\mathbf{sig}}_{(r,v_r)}\right)$ — by running `CryptoAC`.

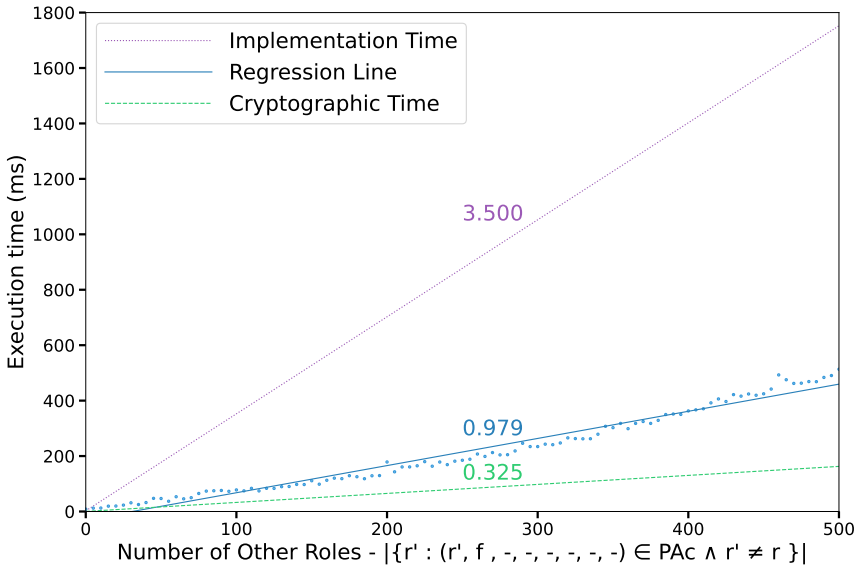


Figure 4.10: Performance of $\underline{revokePermissionFromRole_{adm}(r, \langle f, op \rangle)}$

**Results.** We report the size of each set of $\mathbf{P_t}$ and $\mathbf{P_c}$ in Table 4.10. To give an intuition of the size of the metadata in a realistic scenario, we consider RBAC policy states mined — that is, extracted — by Ene et al. [35] from real-world datasets, which we report in Table 4.11 along with the corresponding size of $\mathbf{P_t}$ and $\mathbf{P_c}$ (computed by simply multiplying the numbers in Tables 4.10 and 4.11 for each set). From the table, we can see that the *emea* state produces the biggest policy size, that is 867KB for $\mathbf{P_t}$ and 3,103KB for $\mathbf{P_c}$; when summed, the overall size of metadata is around 3MB, which we believe to be reasonable in terms of absolute size.

Table 4.10: Size (in bytes) of $\mathbf{P_t}$ and $\mathbf{P_c}$ in our RBAC CAC scheme

| | Symbol | Size (in bytes) |
|---|---|---|
| $\mathbf{P_t}$ | $u \in \mathbf{U_t}$ | 50 |
| | $r \in \mathbf{R_t}$ | 50 |
| | $f \in \mathbf{F_t}$ | 50 |
| | $(u, r) \in \mathbf{UR_t}$ | (50, 50) = 100 |
| | $(r, \langle f, op \rangle) \in \mathbf{PA_t}$ | (50, 50, 1) = 101 |
| $\mathbf{P_c}$ | $(u, \mathbf{p}_u, \mathbf{k}_u^{\mathbf{enc}}, \mathbf{k}_u^{\mathbf{ver}}, \mathbf{S}) \in \mathbf{U_c}$ | (50, 50, 32, 32, 1) = 165 |
| | $(r, \mathbf{p}_{(r,v_r)}, \mathbf{k}_{(r,v_r)}^{\mathbf{enc}}, \mathbf{k}_{(r,v_r)}^{\mathbf{ver}}, v_r, \mathbf{S}) \in \mathbf{R_c}$ | (50, 50, 32, 32, 8, 1) = 173 |
| | $(f, \mathbf{p}_{(f,v_{f\_enc})}, v_{f\_dec}, v_{f\_enc}, \mathbf{S}, \mathbf{Enc}) \in \mathbf{F_c}$ | (50, 50, 8, 8, 1, 1) = 118 |
| | $(u, r, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}_u^{\mathbf{enc}}} \left( \mathbf{k}_{(r,v_r)}^{\mathbf{enc}} \mathbf{k}_{(r,v_r)}^{\mathbf{dec}}, \mathbf{k}_{(r,v_r)}^{\mathbf{ver}}, \mathbf{k}_{(r,v_r)}^{\mathbf{sig}} \right), v_r) \in \mathbf{UR_c}$ | (50, 50, 352, 8) = 460 |
| | $(r, f, \mathbf{p}_{(r,v_r)}, \mathbf{p}_{(f,v_f)}, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}_{(r,v_r)}^{\mathbf{enc}}} \left( \mathbf{k}_{(f,v_{f\_dec})}^{\mathbf{sym}} \right), \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}_{(r,v_r)}^{\mathbf{enc}}} \left( \mathbf{k}_{(f,v_{f\_enc})}^{\mathbf{sym}} \right), v_r, v_f, op) \in \mathbf{PA_c}$ | (50, 50, 50, 50, 80, 80, 8, 8, 1) = 377 |

Table 4.11: RBAC policy states from [35] and corresponding policy size

| State | Set | | | | | Size (in KB) | |
|---|---|---|---|---|---|---|---|
| | $\lvert\mathbf{U}\rvert$ | $\lvert\mathbf{R}\rvert$ | $\lvert\mathbf{F}\rvert$ | $\lvert\mathbf{UR}\rvert$ | $\lvert\mathbf{PA}\rvert$ | $\mathbf{P_t}$ | $\mathbf{P_c}$ |
| *domino* | 79 | 20 | 231 | 75 | 629 | 88 | 315 |
| *emea* | 35 | 34 | 3,046 | 35 | 7,211 | 867 | 3,103 |
| *firewall1* | 365 | 60 | 709 | 1,130 | 3,455 | 506 | 1,977 |
| *firewall2* | 325 | 10 | 590 | 325 | 1,136 | 189 | 703 |
| *healthcare* | 46 | 13 | 46 | 55 | 359 | 46 | 176 |
| *university* | 493 | 16 | 56 | 495 | 202 | 96 | 395 |

## 4.7 Related Work for Our Cryptographic Access Control Schemes

We now analyze other works in the literature proposing CAC schemes for protecting data in transit or at rest. To this end, we cluster these works based on common characteristics and describe them so to identify their differences with respect to our two CAC schemes presented in Sections 4.3 and 4.4.

**Cryptographic Access Control Schemes Relying on Cloud Providers.** Zhang et al. [134] propose a CAC scheme based on CP-ABE featuring the possibility of outsourcing cryptographic computations to the edge and the cloud. This possibility relieves the overhead of CAC on end devices, making the CAC scheme particularly suitable to IoT scenarios. However, at the same time, the authors assume full trust on the cloud provider, an assumption that may limit the applicability of the CAC scheme.

Also Xiao et al. [129] — whose main focus is on the possibility of querying encrypted data — consider outsourced decryption for CP-ABE. Moreover, the authors delegate to edge nodes the revoking of privileges to users, which may therefore collude with honest-but-curious cloud providers for unauthorized access to data.

In [131], the authors consider a generic scenario in which computationally constrained IoT devices generate sensitive data that has to be securely stored in the cloud. In this context, the authors combine CP-ABE and Key-Policy Attribute-Based Signature (KP-ABS) into a single cryptographic primitive, where cryptographic computations such as signature verification and ciphertext decryption are offloaded to edge nodes — which, again, are assumed to be operated by a fully trusted cloud provider.

Fugkeaw and Sato [110] propose a CAC scheme for a hybrid AC model combining RBAC and ABAC. The authors implement their CAC scheme in an administrative tool named CLOUD-CAT logically composed of four modules responsible for users' authentication and authorization, AC policy management and enforcement, and data en/decryption. However, the authors do not discuss the possible presence of honest-but-curious cloud providers.

Summarizing, relying on cloud providers to, e.g., perform cryptographic computations or carry out revocation processes, allows for relieving the computational overhead of CAC and improving its scalability for cloud native applications. However, these benefits must be carefully balanced with the possible presence of honest-but-curious cloud providers which may violate the confidentiality of sensitive data.

**Cryptographic Access Control Schemes with Fixed Placements.** Zhou et al. [135] propose a CAC scheme in which RBAC policies are enforced using a novel cryptographic primitive called Role-Based Encryption (RBE). The CAC scheme expects the presence of both a public and a private cloud: the former stores encrypted data, while the latter stores metadata. Users communicate only with the public cloud, while the administrator interacts with the private cloud. Moreover, the public cloud performs some cryptographic computations on behalf of the users and communicates with the private cloud to retrieve the necessary metadata. Although being an interesting proposal, we note that such a CAC scheme expects a fixed placement of entities to domains which — as thoroughly discussed in Chapter 2 — may not be suitable for any scenario.

Similarly, Zarandioon et al. [133] propose a CAC scheme with a focus on key updating. The authors emphasize privacy with respect to the AC policy and users' operations, enabling anonymous access to encrypted resources at rest in the cloud. Notably, resources are versioned, hence the RM is not a required entity, as discussed in Section 2.2.2. However, the placement of the other entities is fixed and cannot be modified to accommodate other scenarios.

Tang et al. [115] present a CAC scheme with a focus on assured deletion of resources from the cloud and implement their CAC scheme in a proof-of-concept called FADE. The placement of FADE expects a quorum of key managers deployed on-premise as a central trusted entity (i.e., the administrator) and a proxy that can run either on users' devices or on-premise as well.

Finally, Qi and Zheng [95] propose a CAC scheme whose design is inspired by the work in [39]. Differently than [39], the proposed CAC scheme carries out revocations through onion encryption: each time a privilege is revoked from a user, the cloud provider is expected to add an encryption layer with a new key on each affected resource. Intuitively, to read a resource, an authorized user has to decrypt all layers. The administrator can set a threshold on the number of encryption layers, after which a de-onioning procedure occurs. The authors implement their scheme and experimentally measure an overhead of 7.2% with respect to [39] — with the significant advantage of immediately blocking access to resources by revoked users. However, the proposed CAC scheme expects all entities, except the proxy, to be assigned to the cloud.

Summarizing, the aforementioned works propose CAC schemes with interesting features that may suit specific scenarios. However, on the other hand, restricting the possible assignments of entities to domains may limit the applicability of the CAC schemes.

**Cryptographic Access Control Schemes with Static Policies.** Nasiraee et al. [84] foster the use of ABE for enforcing fine-grained AC policies in cloud-based scenarios. However, the authors argue that having a single administrator generating ABE secret keys for users leads to a key escrow issue that mines the robustness and security of the whole system. Therefore, the authors propose a CAC scheme with multiple partially trusted administrators (called "attribute authorities"), each responsible for a disjoint subset of the set of all attributes. The CAC scheme has many interesting features, such as the use of cryptographic accumulators to enhance privacy, the protection of the ABAC policy with obfuscation, and the outsourcing of cryptographic computations. However, the authors do not discuss the dynamicity of the ABAC policy, hence limiting the applicability of their CAC scheme to static scenarios only.

Similarly, the authors in [40] and [136] propose two interesting CAC schemes, the former enforcing RBAC policies using ABE and the latter offering controlled leakage, that is, once a data leakage occurs, the confidentiality of unleaked data is guaranteed. Nonetheless, both CAC schemes expect static AC policies.

In [71], the authors propose a CAC scheme enabling multi-keyword searchable encryption for sensitive resources stored in the cloud. The CAC scheme preserves the privacy of the queries and the keyword indexes while including also mechanisms to verify the accuracy of the results returned by the cloud provider. As the focus of the authors is on querying, the dynamicity of the AC policy — as well as possible collusions between unauthorized users and the cloud provider — is not discussed.

In [59], the author designs a CAC scheme using non-monotonic CP-ABE. Similarly to our CAC schemes, an administrator is responsible for creating ABE secret keys, while users interact with resources through a proxy. Unfortunately, the CAC scheme does not support updating the AC policy.

Finally, in [113] the authors propose a CAC scheme for MQTT based on ABE. For the E2E encryption of data, the authors consider both CP-ABE and KP-ABE. In the CAC scheme, publishers and subscribers register to a trusted authority (i.e., the administrator) to obtain ABE private keys embedding their attributes (in CP-ABE) or their access policy (if KP-ABE). The authors highlight that CP-ABE is more flexible but incurs a greater overhead than KP-ABE, hence CP-ABE may not be suitable for resource-constrained IoT devices. On the other hand, KP-ABE requires either constant interaction between MQTT clients and the trusted authority or the definition of keys and a standard set of AC policies a-priori. The CAC scheme provides E2E encryption, AC enforcement and exhibits

great scalability. However, the dynamicity of the AC policy (e.g., the re-distribution of cryptographic material after the revocation of a permission) is not discussed, nor is the integrity of messages.

Summarizing, the aforementioned works all propose remarkable designs for enforcing AC policies through cryptography, although the lack of the possibility to update AC policies might hinder the versatility of these CAC schemes.

# Chapter 5

# Conclusion

Data security in cloud native applications is a complex matter that requires addressing numerous challenges. In this thesis, we propose a security service that tackles some of these challenges — in a nutshell, zero trust AC, confidentiality and integrity, and security-performance balancing — by providing (*i*) privileged access management and data protection in the Cloud-to-Thing Continuum together with (*ii*) the ability to evaluate its performance under realistic workloads and (*iii*) the possibility to optimally integrate it into cloud native applications. Our security service comprises four security mechanisms: `CryptoAC` , `ACE` and `ACME`, and `MOMO`, which implement the contributions of our thesis.

`CryptoAC` (*i* above) implements two CAC schemes, one for RBAC and one for ABAC, providing E2E protection and fine-grained AC over resources in transit and at rest in cloud native applications. The microservice-based design and modularity of `CryptoAC` (Section 4.5) allows for seamless integration with cloud native applications. Moreover, the experimental evaluation conducted on the RBAC CAC scheme implemented in `CryptoAC` (Section 4.6) highlights reasonable — although potentially improvable — performance and, especially, a natural scalability thanks to the distribution of cryptographic computations to end devices.

`ACE` and `ACME` (*ii* above) implement the two modules of the methodology for measuring the performance of AC enforcement mechanisms, i.e., the procedure deriving sequences of AC requests from BPMN workflows and the evaluator executing them on different mechanisms, respectively. The proof-of-concept application on OPA, XACML, and `CryptoAC` show that our methodology provides a robust basis for realistic performance evaluation, being even able to capture the benefits — in terms of scalability — of decentralized AC enforcement mechanisms (Section 3.5). Moreover, we demonstrate how micro-benchmarks, although definitively useful, cannot return realistic performance evaluation results and may therefore not always suggest the best mechanism for a given scenario (Section 3.5.4).

`MOMO` (*iii* above) implement the (algorithm to solve the) MOCOP allowing for identifying the placement of microservices composing CAC enforcement mechanisms that simultaneously optimize a set of (possibly conflicting) security and performance objectives specific to cloud native applications. The conceptual application to the remote patient monitoring scenario showcases the benefits of our MOCOP at the theoretical level (Section 2.4.1), while the proof-of-concept application to the CLC service demonstrates such benefits at the concrete level — that is, when integrating `MOMO` as a placement algorithm for K8s in FogAtlas (Section 2.5).

## 5.1 Future Work and Research Directions

Below, we describe our future work and possible research directions on (the contributions implemented by) MOMO (Section 5.1.1), ACE and ACME (Section 5.1.2), and CryptoAC (Section 5.1.3).

### 5.1.1 Generalizing the Applicability of the Optimization Problem

Natural future work on MOMO (Section 2.5) consists in refining its implementation and integration with FogAtlas and K8s, as well as offering administrators the means (e.g., a dashboard) to easily configure all of the parameters of the MOCOP presented in Section 2.4. Then, we note that our MOCOP is tailored for a specific security mechanism, i.e., CAC. However, cloud native applications rely on a wide range of security mechanisms offering functions such as network traffic filtering and inspection (e.g., firewalls, deep packet inspection), anomaly detection (e.g., intrusion detection systems), and attack deception (e.g., Honeypot). Nonetheless, despite providing different functions, the balancing of security and performance — extensively discussed in Chapter 2 — is a common concern that affects both CAC and these security mechanisms. Therefore, we are working on generalizing our MOCOP to expand its scope and applicability for generic security mechanisms and cloud native applications.

### 5.1.2 Extending the Capabilities and Scope of Our Methodology

Natural improvements to our methodology are a more fine-grained customization of the evaluation (e.g., tuning default activities input/output) and the implementation of further adapters for ACME. Moreover, we are planning to expand the set of BPMN symbols supported by ACE and parse more workflows, hence creating a library of workflows that organizations (as well as researchers) can readily use to evaluate the performance of their AC enforcement mechanisms — acting therefore as a reference benchmarking solution. Another interesting research direction is extending our methodology to support the evaluation of ABAC enforcement mechanisms as well. At the conceptual level, we note that ACE could support ABAC already, except for the derivation of state-change rules for ABAC from WF net executions (step 4 in Figure 3.3). Finally, it would be interesting to correlate the idling time of workflows (discussed in Section 3.4.2) with the concept of AC policy resiliency [70] and, more in general, the workflow satisfiability problem [124]. In this regard, we could also include the specification of (dynamic and static) separation of duty constraints [67] into ACE by, e.g., introducing ad-hoc state-change rules. For instance, given two permissions $p_1$ and $p_2$ assigned to two roles $r_1$ and $r_2$, respectively, and a separation of duty constraint saying that the same user $u$ cannot assume $r_2$ to execute $p_2$ if $u$ already assumed $r_1$ to execute $p_1$, we could introduce a special state-change rule that revokes $u$ from $r_2$ whenever $u$ executes $p_1$ using $r_1$.

### 5.1.3 Improving Security and Performance of Cryptographic Access Control Schemes

Natural follow-ups to our work on CAC are a systematization of the design of the two CAC schemes, the formal demonstration of their security (e.g., in the symbolic model), and a thorough performance evaluation for the ABAC CAC scheme. Then, we note that, despite holding the potential to become a valid alternative to other security mechanisms for data protection and AC in cloud native applications, CryptoAC is still a proof-of-concept, and a concrete deployment should consider additional aspects such as, e.g., key management and users' authentication (e.g., through OpenID connect[1]). Therefore, we are studying the integration of CryptoAC with security mechanisms addressing these aspects. Besides, although the current design of CryptoAC already aims at minimizing the computational

[1] OpenID (https://openid.net/)

overhead of CAC (Section 4.2), we believe that performance may be further improved. In this regard, we are investigating two research directions. First, we are working on an extended RBAC model specifically tailored for CAC to enhance its expressiveness and enhance the hybrid enforcement (see Section 4.2.4). More in detail, the extended RBAC model will allow administrators to specify different trust levels on users, roles, and resources based on which fine-tune the cryptographic computations expected by CAC. Secondly, we are exploring the possible synergies between CAC and TEEs. In particular, the use of TEEs may allow for concealing private and secret keys from users, hence streamlining the operations concerning the revoking of privileges from users — which are the most computationally demanding operations, as shown in Section 4.6.3. As a final remark, our CAC schemes target data in transit and at rest; intuitively, data may also need to be protected when in use (i.e., when being processed). To this end, we note that there already exist several cryptographic-based solutions to protect the confidentiality of data and the processing itself, such as homomorphic encryption and functional encryption [43]. In this context, we believe CAC to be complementary — and not opposite — to such solutions. For instance, data may be encrypted with functional encryption, and the secret keys allowing to compute functions over the encrypted data (i.e., the secret functional encryption keys) may be distributed through CAC.

# Bibliography


[1] A. Abbas and S. 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] T. Ahmad, U. Morelli, S. Ranise, and N. Zannone. Extending access control in AWS IoT through event-driven functions: an experimental evaluation using a smart lock system. *International Journal of Information Security*, 2021.

[3] S. Ahmadi, M. Nassiri, and M. Rezvani. XACBench: a XACML policy benchmark. *Soft Computing*, 24(21):16081–16096, 2020.

[4] N. Alshuqayran, N. Ali, and R. Evans. A systematic mapping study in microservice architecture. In *2016 IEEE 9th InternatConference on Service-Oriented Computing and Applications (SOCA)*, pages 44–51. IEEE, 2016.

[5] Y. Ashibani and Q. H. Mahmoud. Cyber physical systems security: Analysis, challenges and solutions. *Computers & Security*, 68:81–97, 2017.

[6] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, 2008.

[7] S. Berlato, R. Carbone, A. J. Lee, and S. Ranise. Exploring architectures for cryptographic access control enforcement in the cloud for fun and optimization. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 208–221. ACM, 2020.

[8] S. Berlato, R. Carbone, A. J. Lee, and S. Ranise. Formal modelling and automated trade-off analysis of enforcement architectures for cryptographic access control in the cloud. *ACM Trans. Priv. Secur.*, 25(1), nov 2021.

[9] S. Berlato, R. Carbone, and S. Ranise. Cryptographic enforcement of access control policies in the cloud: Implementation and experimental assessment. In S. D. C. di Vimercati and P. Samarati, editors, *Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021, July 6-8, 2021*, pages 370–381. SCITEPRESS, 2021.

[10] S. Berlato, M. Centenaro, and S. Ranise. Smart card-based identity management protocols for v2v and v2i communications in CCAM: A systematic literature review. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):10086–10103, 2022.

[11] S. Berlato, U. Morelli, R. Carbone, and S. Ranise. End-to-end protection of IoT communications through cryptographic enforcement of access control policies. In S. Sural and H. Lu, editors, *Data and Applications Security and Privacy XXXVI*, volume 13383, pages 236–255. Springer International Publishing, 2022. Series Title: Lecture Notes in Computer Science.

[12] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, pages 321–334, 2007.

[13] J. Bonneau and I. Mironov. Cache-collision timing attacks against AES. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249, pages 201–215. Springer Berlin Heidelberg, 2006. Series Title: Lecture Notes in Computer Science.

[14] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[15] A. D. Brucker. Integrating security aspects into business process models. *it - Information Technology*, 55(6):239–246, 2013.

[16] B. Butler and B. Jennings. Measurement and prediction of access control policy evaluation performance. *IEEE Transactions on Network and Service Management*, 12(4):526–539, 2015.

[17] B. Butler, B. Jennings, and D. Botvich. Xacml policy performance evaluation using a flexible load testing framework. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, page 648–650, New York, NY, USA, 2010. Association for Computing Machinery.

[18] B. Butler, B. Jennings, and D. Botvich. An experimental testbed to predict the performance of XACML policy decision points. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 353–360. IEEE, 2011.

[19] F. Cai, N. Zhu, J. He, P. Mu, W. Li, and Y. Yu. Survey of access control models and technologies for cloud computing. *Cluster Computing*, 22:6111–6122, 2019.

[20] R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In L. R. Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002*, volume 2332, pages 337–351. Springer Berlin Heidelberg, May 2002.

[21] M. Centenaro, S. Berlato, R. Carbone, G. Burzio, G. F. Cordella, S. Ranise, and R. Riggio. Security considerations on 5g-enabled back-situation awareness for CCAM. In *2020 IEEE 3rd 5G World Forum (5GWF)*, pages 245–250. IEEE, 2020.

[22] M. Centenaro, S. Berlato, R. Carbone, G. Burzio, G. F. Cordella, R. Riggio, and S. Ranise. Safety-related cooperative, connected, and automated mobility services: Interplay between functional and security requirements. *IEEE Vehicular Technology Magazine*, 16(4):78–88, 2021.

[23] P. Chandrakar, A. Jain, S. Balivada, and R. Ali. A secure authentication protocol for vehicular ad-hoc networks. In *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–7. IEEE, Feb. 2019.

[24] G.-C. Consortium. Design of the secure, cross-border, and multi-domain service orchestration platform, 2020.

[25] G.-C. Consortium. Results on the validation of the 5gcarmen platform, 2922.

[26] H. Cormen, C. Leiserson, R. Rivest, and C. Stein. Topological Sort, Introduction to Algorithms, 2009.

[27] V. Cristiano. Key Management for Cryptographic Enforcement of Access Control Policies in the Cloud: The CryptoAC use case. Master's thesis, University of Trento, Trento, Italy, 2021.

[28] R. M. Dijkman, M. Dumas, and C. Ouyang. Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12):1281–1294, 2008.

[29] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali. Cloud Computing: Distributed Internet Computing for IT and Scientific Research. *IEEE Internet Computing*, 13(5):10–13, Sept. 2009.

[30] J. B. Djoko, J. Lange, and A. 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, 2019.

[31] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, Mar. 1983.

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

[33] D. Domingos, A. R. Silva, and P. Veiga. Workflow access control from a business perspective. In *Proceedings of the 4th International Workshop on Pattern Recognition in Information Systems*, pages 18–25. SciTePress - Science and and Technology Publications, 2004.

[34] B. S. Egala, A. K. Pradhan, V. Badarla, and S. P. Mohanty. Fortified-chain: A blockchain-based framework for security and privacy-assured internet of medical things with effective access control. *IEEE Internet of Things Journal*, 8(14):11717–11731, 2021.

[35] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *Proceedings of the 13th ACM symposium on Access control models and technologies - SACMAT '08*, page 1. ACM Press, 2008.

[36] A. L. Ferrara, G. Fachsbauer, B. Liu, and B. Warinschi. Policy Privacy in Cryptographic Access Control. In *2015 IEEE 28th Computer Security Foundations Symposium*, pages 46–60, Verona, July 2015. IEEE.

[37] A. N. S. for Information Technology. Role based access control. Standard, American National Standards Institute, Inc., Feb. 2004.

[38] Gang Ma, Kehe Wu, Tong Zhang, and Wei Li. A flexible policy-based access control model for workflow management systems. In *2011 IEEE International Conference on Computer Science and Automation Engineering*, pages 533–537. IEEE, 2011.

[39] W. C. Garrison, A. Shull, S. Myers, and A. J. Lee. On the practicality of cryptographically enforcing dynamic access control policies in the cloud. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 819–838. IEEE, 2016. event-place: San Jose, CA.

[40] V. Ghita, S. Costea, and N. Tapus. Implementation of cryptographically enforced rbac. *The Scientific Bulletin - University Politehnica of Bucharest*, 79(2):9–3–102, 2017.

[41] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *The VLDB Journal*, 16:5–28, 01 2007.

[42] S. Goel and V. Chen. Information security risk analysis — a matrix-based approach, 2005. Accessed: 2021-09-08.

[43] C. Gottel, R. Pires, I. Rocha, S. Vaucher, P. Felber, M. Pasin, and V. Schiavoni. Security, performance and energy trade-offs of hardware-assisted memory protection mechanisms. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 133–142. IEEE, 2018.

[44] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 89–98, New York, NY, USA, 2006. Association for Computing Machinery.

[45] L. Griffin, B. Butler, E. d. Leastar, B. Jennings, and D. Botvich. On the performance of access control policy evaluation. In *2012 IEEE International Symposium on Policies for Distributed Systems and Networks*, pages 25–32. IEEE, 2012.

[46] O. M. Group. Bpmn 2.0 by example - version 1.0. Non-normative, Object Management Group, June 2010.

[47] O. M. Group. Business process model and notation (bpmn) - version 2.0.2. Standard, Object Management Group, Dec. 2013.

[48] H. W. Hamacher, C. R. Pedersen, and S. Ruzika. Multiple objective minimum cost flow problems: A review. *European Journal of Operational Research*, 176(3):1404–1422, Feb. 2007.

[49] A. Hannousse and S. Yahiouche. Securing microservices and microservice architectures: A systematic mapping study. *Computer Science Review*, 41:100415, 2021.

[50] T. Heer, O. Garcia-Morchon, R. Hummen, S. L. Keoh, S. S. Kumar, and K. Wehrle. Security challenges in the IP-based internet of things. *Wireless Personal Communications*, 61(3):527–542, 2011.

[51] F. Horandner, S. Krenn, A. Migliavacca, F. Thiemer, and B. 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, Aug. 2016. IEEE.

[52] J. Horwitz and B. Lynn. Toward hierarchical identity-based encryption. In L. R. Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002*, volume 2332, pages 466–481. Springer Berlin Heidelberg, 2002. Series Title: Lecture Notes in Computer Science.

[53] J. Horwitz and B. 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.

[54] V. C. Hu, D. F. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. Guide to attribute based access control (abac) definition and considerations, 2014.

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

[56] O. M. Ilhan, D. Thatmann, and A. Kupper. A performance analysis of the XACML decision process and the impact of caching. In *2015 11th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, pages 216–223. IEEE, 2015.

[57] E. T. S. Institute. ETSI TS ITS 102 731; security; security services and architecture; release 2, 2022.

[58] Y. Jadeja and K. 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, Mar. 2012. IEEE.

[59] J. 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), Jan. 2018.

[60] C. Kady, A. M. Chedid, I. Kortbawi, C. Yaacoub, A. Akl, N. Daclin, F. Trousset, F. Pfister, and G. Zacharewicz. IoT-driven workflows for risk management and control of beehives. *Diversity*, 13(7):296, 2021.

[61] A. A. Kalenkova, W. M. P. Van Der Aalst, I. A. Lomazova, and V. A. Rubin. Process mining using BPMN: relating event logs and process models. *Software & Systems Modeling*, 16(4):1019–1048, 2017.

[62] I. Karabey Aksakalli, T. Celik, A. B. Can, and B. Tekinerdogan. Deployment and communication patterns in microservice architectures: A systematic literature review. *Journal of Systems and Software*, 180:111014, 2021.

[63] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

[64] M. T. Khorshed, A. S. Ali, and S. 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.

[65] K. Klamroth. Discrete multiobjective optimization. In M. Ehrgott, C. M. Fonseca, X. Gandibleux, J.-K. Hao, and M. Sevaux, editors, *Evolutionary Multi-Criterion Optimization*, volume 5467, pages 4–4. Springer Berlin Heidelberg, 2009. Series Title: Lecture Notes in Computer Science.

[66] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[67] D. R. Kuhn, E. J. Coyne, and T. R. Weil. Adding attributes to role-based access control. *Computer*, 43(6):79–81, 2010.

[68] R. Kumar and R. Goyal. On cloud security requirements, threats, vulnerabilities and countermeasures: A survey. *Computer Science Review*, 33:1–48, Aug. 2019.

[69] N. Li and M. V. Tripunitara. Security analysis in role-based access control. *ACM Trans. Inf. Syst. Secur.*, 9(4):391–420, nov 2006.

[70] N. Li, Q. Wang, and M. Tripunitara. Resiliency policies in access control. *ACM Transactions on Information and System Security*, 12(4):1–34, 2009.

[71] Y. Liang, Y. Li, Q. Cao, and F. Ren. Vpams: Verifiable and practical attribute-based multi-keyword search over encrypted cloud data. *Journal of Systems Architecture*, 108:101741, 2020.

[72] F. Lins, J. Damasceno, R. Medeiros, E. Sousa, and N. Rosa. Automation of service-based security-aware business processes in the cloud. *Computing*, 98:847–870, 09 2016.

[73] T. Loruenser, D. Slamanig, T. Langer, and H. 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, Aug. 2016. IEEE.

[74] T. Lynn, J. G. Mooney, B. Lee, and P. T. Endo. The cloud-to-thing continuum: Opportunities and challenges in cloud, fog and edge computing. In *The Cloud-to-Thing Continuum*, Palgrave Studies in Digital Business & Enabling Technologies. Springer International Publishing, 2020.

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

[76] F. Martinelli, O. Osliak, P. Mori, and A. Saracino. Improving security in industry 4.0 by extending OPC-UA with usage control. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pages 1–10. ACM, 2020.

[77] F. Martins and D. Domingos. Modelling IoT behaviour within BPMN business processes. *Procedia Computer Science*, 121:1014–1022, 2017.

[78] P. Masek and T. B. Magnus. Container Based Virtualisation for Software Deployment in Self-Driving Vehicles. Master's thesis, Chalmers University of Technology, Gothenburg, Sweden, 2016.

[79] P. M. Mell and T. Grance. The NIST definition of cloud computing, 2011. Edition: 0.

[80] V. S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, volume 218, pages 417–426. Springer Berlin Heidelberg, 1986.

[81] K. Mohammad Hossein, M. E. Esmaeili, T. Dargahi, A. Khonsari, and M. Conti. BCHealth: A novel blockchain-based privacy-preserving architecture for IoT healthcare applications. *Computer Communications*, 180:31–47, 2021.

[82] R. Morabito, R. Petrolo, V. Loscri, N. Mitton, G. Ruggeri, and A. Molinaro. Lightweight virtualization as enabling technology for future smart cars. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 1238–1245. IEEE, 2017.

[83] J. Mülle, S. von Stackelberg, and K. Böhm. A security language for bpmn process models. Technical Report 9, Karlsruher Institut für Technologie (KIT), 2011.

[84] H. Nasiraee and M. Ashouri-Talouki. Anonymous decentralized attribute-based access control for cloud-assisted IoT. *Future Generation Computer Systems*, 110:45–56, 2020.

[85] P. K. P, S. K. P, and A. 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, Apr. 2018.

[86] P. K. P, S. K. P, and A. 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, 2018.

[87] B. Palaniswamy, S. Camtepe, E. Foo, L. Simpson, M. A. Rezazadeh Baee, and J. Pieprzyk. Continuous authentication for vanet. *Vehicular Communications*, 25:100255, Oct. 2020.

[88] A. Palmieri, P. Prem, S. Ranise, U. Morelli, and T. Ahmad. MQTTSA: A tool for automatically assisting the secure deployments of MQTT brokers. In *2019 IEEE World Congress on Services (SERVICES)*, pages 47–53. IEEE, 2019.

[89] F. Palumbo, G. Aceto, A. Botta, D. Ciuonzo, V. Persico, and A. Pescapé. Characterization and analysis of cloud-to-user latency: The case of azure and AWS. *Computer Networks*, 184:107693, 2021.

[90] A. A.-N. Patwary, A. Fu, R. K. Naha, S. K. Battula, S. Garg, M. A. K. Patwary, and E. Aghasian. Authentication, access control, privacy, threats and trust management towards securing fog computing environments: A review, 2020.

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

[92] A. Pereira-Vale, E. B. Fernandez, R. Monge, H. Astudillo, and G. Márquez. Security in microservice-based systems: A multivocal literature review. *Computers & Security*, 103:102200, 2021.

[93] J. Petit, F. Schaub, M. Feiri, and F. Kargl. Pseudonym schemes in vehicular networks: A survey. *IEEE Communications Surveys & Tutorials*, 17(1):228–255, 2015.

[94] U. Premarathne, A. Abuadbba, A. Alabdulatif, I. Khalil, Z. Tari, A. Zomaya, and R. Buyya. Hybrid Cryptographic Access Control for Cloud-Based EHR Systems. *IEEE Cloud Computing*, 3(4):58–64, July 2016.

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

[96] G. Ramachandra, M. Iftikhar, and F. A. Khan. A Comprehensive Survey on Security in Cloud Computing. *Procedia Computer Science*, 110:465–472, 2017.

[97] E. Ramirez, J. Brill, M. Ohlhausen, J. 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, Jan. 2014.

[98] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3, Aug. 2018.

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

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

[101] S. Rose, O. Borchert, S. Mitchell, and S. Connelly. Zero trust architecture, 2020.

[102] C. Ruland and J. Sassmannshausen. Access control in safety critical environments. In *2018 12th International Conference on Reliability, Maintainability, and Safety (ICRMS)*, pages 223–229. IEEE, 2018.

[103] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *Advances in Cryptology – EUROCRYPT 2005*, volume 3494, pages 457–473. Springer Berlin Heidelberg, 2005.

[104] M. Salnitri, A. Brucker, and P. Giorgini. From secure business process models to secure artifact-centric specifications. *Lecture Notes in Business Information Processing*, 214:246–262, 06 2015.

[105] M. Salnitri, F. Dalpiaz, and P. Giorgini. Modeling and verifying security policies in business processes. In I. Bider, K. Gaaloul, J. Krogstie, S. Nurcan, H. A. Proper, R. Schmidt, and P. Soffer, editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 175, pages 200–214. Springer Berlin Heidelberg, 2014. Series Title: Lecture Notes in Business Information Processing.

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

[107] R. Sandhu and P. Samarati. Access control: principle and practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.

[108] E. B. Sanjuan, I. A. Cardiel, J. A. Cerrada, and C. Cerrada. Message queuing telemetry transport (MQTT) security: A cryptographic smart card approach. *IEEE Access*, 8:115051–115062, 2020.

[109] H. Sato and S. 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, Sept. 2015.

[110] H. Sato and S. 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, Sept. 2015.

[111] D. Servos and S. L. Osborn. Current research and open problems in attribute-based access control. *ACM Computing Surveys*, 49(4):1–45, 2017.

[112] A. Singh and K. Chatterjee. Cloud security issues and challenges: A survey. *Journal of Network and Computer Applications*, 79:88–115, Feb. 2017.

[113] M. Singh, M. Rajan, V. Shivraj, and P. Balamuralidhar. Secure MQTT for internet of things (IoT). In *2015 Fifth International Conference on Communication Systems and Network Technologies*, pages 746–751. IEEE, 2015.

[114] D. Strom and J. F. van der Zwet. Truth and lies about latency in the cloud. *InterxionTM white paper*, 2012.

[115] Y. Tang, P. P. C. Lee, J. C. S. Lui, and R. Perlman. FADE: Secure overlay cloud storage with file assured deletion. In S. Jajodia and J. Zhou, editors, *Security and Privacy in Communication Networks*, volume 50, pages 380–397. Springer Berlin Heidelberg, 2010. Series Title: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering.

[116] F. L. Tiplea, C. Bocaneala, and R. Chirosca. On the complexity of deciding soundness of acyclic workflow nets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(9):1292–1298, 2015.

[117] S. H. Turki, F. Bellaaj, A. Charfi, and R. Bouaziz. Modeling security requirements in service based business processes. In I. Bider, T. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, and S. Wrycza, editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 113, pages 76–90. Springer Berlin Heidelberg, 2012. Series Title: Lecture Notes in Business Information Processing.

[118] F. Turkmen and B. Crispo. Performance evaluation of XACML PDP implementations. In *Proceedings of the 2008 ACM workshop on Secure web services - SWS '08*, page 37. ACM Press, 2008.

[119] M. Uddin, S. Islam, and A. Al-Nemrat. A dynamic access control model using authorising workflow and task-role-based access control. *IEEE Access*, 7:166676–166689, 2019.

[120] W. M. Van der Aalst. Structural characterizations of sound workflow nets. *Computing science reports*, 96(23):18–22, 1996.

[121] W. M. van der Aalst and A. H. ter Hofstede. Verification of workflow task structures: A petri-net-baset approach. *Information Systems*, 25(1):43–69, 2000.

[122] M. Venema, G. Alpár, and J.-H. Hoepman. Systematizing core properties of pairing-based attribute-based encryption to uncover remaining challenges in enforcing access control in practice. *Designs, Codes and Cryptography*, 91(1):165–220, 2023.

[123] J. Wainer, P. Barthelmess, and A. Kumar. W-RBAC — a workflow security model incorporating controlled overriding of constraints. *International Journal of Cooperative Information Systems*, 12(4):455–485, 2003.

[124] Q. Wang and N. Li. Satisfiability and resiliency in workflow systems. In J. Biskup and J. López, editors, *Computer Security – ESORICS 2007*, volume 4734, pages 90–105. Springer Berlin Heidelberg, 2007. Series Title: Lecture Notes in Computer Science.

[125] Y. Wang and Q. Bao. Adapting a container infrastructure for autonomous vehicle development. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0182–0187. IEEE, 2020.

[126] B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, editors, *Public Key Cryptography – PKC 2011*, volume 6571, pages 53–70. Springer Berlin Heidelberg, 2011. Series Title: Lecture Notes in Computer Science.

[127] M. Wazid, A. K. Das, R. Hussain, G. Succi, and J. J. Rodrigues. Authentication in cloud-driven IoT-based big data environment: Survey and outlook. *Journal of Systems Architecture*, 97:185–196, Aug. 2019.

[128] Wei Xu, Jun Wei, Yu Liu, and Jing Li. SOWAC: a service-oriented workflow access control model. In *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, pages 128–134. IEEE, 2004.

[129] M. Xiao, J. Zhou, X. Liu, and M. Jiang. A hybrid scheme for fine-grained search and access authorization in fog computing environment. *Sensors*, 17(6):1423, 2017.

[130] T. Yarygina and A. H. Bagge. Overcoming security challenges in microservice architectures. In *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 11–20. IEEE, 2018.

[131] J. Yu, S. Liu, S. Wang, Y. Xiao, and B. Yan. LH-ABSC: A lightweight hybrid attribute-based signcryption scheme for cloud-fog-assisted IoT. *IEEE Internet of Things Journal*, 7(9):7949–7966, 2020.

[132] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang. A survey on the edge computing for the internet of things. *IEEE Access*, 6:6900–6919, 2018.

[133] S. Zarandioon, D. Yao, and V. Ganapathy. K2c: Cryptographic Cloud Storage with Lazy Revocation and Anonymous Access. In M. Rajarajan, F. Piper, H. Wang, and G. Kesidis, editors, *Security and Privacy in Communication Networks*, volume 96, pages 59–76. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[134] P. Zhang, Z. Chen, J. K. Liu, K. Liang, and H. Liu. An efficient access control scheme with outsourcing capability and attribute update for fog computing. *Future Generation Computer Systems*, 78:753–762, 2018.

[135] L. Zhou, V. Varadharajan, and M. Hitchens. Achieving Secure Role-Based Access Control on Encrypted Data in Cloud Storage. *IEEE Transactions on Information Forensics and Security*, 8(12):1947–1960, Dec. 2013.

[136] X. Zhou, J. Liu, Z. Zhang, and Q. Wu. Secure outsourced medical data against unexpected leakage with flexible access control in a cloud storage system. *Security and Communication Networks*, 2020:1–20, 2020.

[137] C. Zuo, Z. Lin, and Y. Zhang. Why does your data leak? uncovering the data leakage in cloud from mobile apps. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1296–1310. IEEE, 2019.