A Secure and Quality of Service-Aware Solution for the End-to-End Protection of IoT Applications

Stefano Berlato[a,b], Umberto Morelli[b], Roberto Carbone[b], Silvio Ranise[c,b]

[a] *DIBRIS, University of Genova, Genoa, Italy*
[b] *Center for Cybersecurity, Fondazione Bruno Kessler, Trento, Italy*
[c] *Department of Mathematics, University of Trento, Trento, Italy*

Abstract

Internet of Things (IoT) applications increasingly rely on lightweight publish-subscribe protocols (e.g., MQTT) to exchange a considerable amount of sensitive data. However, such data are often threatened by external attackers, malicious insiders, and *honest but curious* Edge and Cloud providers. Typical security mechanisms — such as Transport Layer Security (TLS) or centralized data authorization management — may expose messages to intermediate nodes and fail to enforce Access Control (AC) policies without relying on (sometimes missing) fully trusted agents. Furthermore, when security mechanisms are in place, they should consider the trust assumptions (e.g., on the presence of certain attackers) and meet the performance goals (e.g., low latency, high scalability) relevant to the underlying scenario. In this paper, we propose a security mechanism based Cryptographic Access Control (CAC) that integrates decentralized AC enforcement with end-to-end protection (in terms of data confidentiality and integrity) for IoT applications employing publish-subscribe protocols. By building on previous work, we also formalize an optimization problem to strike the best possible balance between security and quality of service by fine-tuning the deployment of our security mechanism accordingly. We showcase the benefits of the optimization problem in three different scenarios for IoT applications: Remote Patient Monitoring, Cooperative Maneuvering, and Smart Lock. Finally, our open-source proof-of-concept named CryptoAC demonstrates the feasibility of our security mechanism: a thorough performance evaluation reveals that CryptoAC achieves higher scalability than TLS under multi-publisher workloads and a practical level of overhead for key management and policy updates.

Keywords: Cryptographic Access Control, Internet of Things, End-to-end Protection, MQTT

1. Introduction

Field of Research. The capillary diffusion of Internet of Things (IoT) devices, in combination with Edge and Cloud computing, holds the potential to improve the well-being of society in several application fields, like eHealth, intelligent transportation systems, and smart buildings. However, the undeniable benefits offered by IoT should be coupled with appropriate security mechanisms. Indeed, the environment in which IoT applications are deployed is traditionally assumed to be hostile due to the presence of external attackers. Moreover, IoT devices are intrinsically vulnerable and exposed to a wide array of threats, as they are typically left unattended, equipped with limited computational power, and often insecure by design [50]. Hence, suitable security mechanisms should be adopted to ensure the protection of sensitive (e.g., personal or confidential) data throughout their life cycle, namely when in transit, in use, and at rest. In particular, IoT applications often focus on data transmission, which is one of the fundamental layers of their architectures [68]. In this context, protecting communications and encrypting sensitive data are among the top security concerns in IoT.[1] In this regard, since the traditional client-server network paradigm does not properly fit the peculiarities of IoT applications (e.g., unreliable channels), these usually employ more lightweight and efficient *topic-based publish-subscribe* protocols such as Message Queue Telemetry Transport (MQTT) and Advanced Message Queuing Protocol (AMQP) [59, 62].

A Multi-Faceted Problem. However, data protection in IoT communication has many facets. First, the complexity (e.g., large attack surface) and dynamicity of IoT applications — particularly given recent security paradigms such as the *Zero Trust* model — make it difficult to fully trust any involved agent. Instead, agents (e.g., Cloud providers) are usually assumed to be untrusted or partially trusted (i.e., *honest but curious*), meaning they will faithfully carry out assigned tasks but, at the same time, attempt to access sensitive data for purposes such as profiling or profit [34, 12]. For instance, the US Federal Trade Commission[2] fined Ring $5.8 million in May 2023 for compromising customer privacy by allowing employees and contractors to access users' private videos; similarly, Gravy Analytics in January 2025 leaked 10 Terabytes of personal data (e.g., locations) aggregated from popular applications such as Tinder,

[1] `https://outreach.eclipse.foundation/iot-edge-developer-survey-2022`

[2] `https://www.ftc.gov/news-events/news/press-releases/2023/05/ftc-says-ring-employees-illegally-surveilled-customers-failed-stop-hackers-taking-control-users`

IoT camera management and pregnancy support services.[3] In other words, besides external attackers, sensitive data in IoT communication must also be secured from malicious insiders (i.e., legitimate participants who become compromised or malicious, such as disgruntled employees and harmful tenants) and honest but curious Cloud and Edge providers.

The use of cryptography is therefore fundamental to mitigate and prevent possible attacks to the confidentiality and the integrity of data. Indeed, these two security properties are especially important in scenarios involving personal information (e.g., users' health data) or providing vital services where data integrity is critical (e.g., smart smoke sensors). A popular cryptographic mechanism for protecting communications is the Transport Layer Security (TLS) protocol. However, TLS offers hop-to-hop protection only (e.g., between an IoT device and the MQTT broker), leaving data in plaintext at intermediate nodes. In other words, TLS is unable to protect sensitive data from partially trusted agents mediating the communication (e.g., an Edge provider operating an MQTT broker). More generally, TLS does not allow for defining the intended recipient(s) of the data, which is implicitly assumed to be the agent at the other end of the channel — that is, it is not possible to specify Access Control (AC) policies directly within TLS. A similar reasoning also applies to other common cryptographic mechanisms for communications such as Virtual Private Networks (VPNs) and Secure Shell Protocol (SSH) tunneling. Cloud providers managing MQTT brokers and MQTT brokers themselves can enforce AC policies, e.g., see the AWS IoT Core Policies service[4] and the DYNamic SECurity (DYNSEC) plugin[5] — which we also discuss in Section 3.2. However, Cloud- and broker-based AC enforcement mechanisms cannot prevent insiders or partially trusted Cloud providers from viewing sensitive data.

Then, the architecture of cryptographic mechanisms should be capable of adapting to the needs — in terms of both security and quality of service — of different scenarios. Indeed, IoT applications are deployed across a wide variety of different scenarios, each with distinct trust assumptions (e.g., on the presence of certain attackers) and performance goals (e.g., low latency, high scalability). Besides, these assumptions and goals may conflict, requiring a careful analysis of compromises and trade-offs. For instance, processing data in the Cloud promotes reliability and resilience against Denial of Service (DoS) attacks, but it also increases latency with respect to the Edge. At the same time, offloading cryptographic computations to Edge nodes relieves the computational burden on (constrained) IoT devices, but it also risks exposure of sensitive data to malicious insiders or curious tenants. Summarizing, channel protection in IoT is less effective if enforced without knowing the intended recipients (i.e., AC) and fine-tuning security mechanisms for satisfying the (possibly conflicting) security and quality of service requirements of the underlying scenario.

In conclusion, while existing works have explored TLS-based encryption or broker-centric AC enforcement (as also discussed later in Section 2), few solutions cryptographically tie AC policies directly to the data, leaving a gap in scenarios where no agent is fully trusted.

Proposed Solution. In this paper, we address the aforementioned issues by proposing a solution for the end-to-end protection of sensitive data exchanged in IoT applications through the cryptographic enforcement of AC policies — a technique usually called Cryptographic Access Control (CAC). In other words, our solution considers a security mechanism that integrates data confidentiality and integrity guarantees with enforcement of AC policies and optimization of quality of service — the latter being essential for adapting the security mechanism to the peculiarities of different scenarios on a case-by-case basis.

Key Contributions. Our contributions are as follows:

- we design a CAC scheme enforcing Role-Based Access Control (RBAC) policies in IoT applications employing topic-based publish-subscribe protocols (e.g., MQTT), protecting confidentiality and the integrity of sensitive data against external attackers, malicious insiders, and partially trusted agents (Section 6);
- we formalize an optimization problem to fine-tune the architecture of our CAC scheme to strike the best possible balance between security and quality of service according to the trust assumptions and performance goals of a given scenario (Section 7);
- we provide a proof-of-concept application of our optimization problem on three different scenarios, i.e., Remote Patient Monitoring, Cooperative Maneuvering, and Smart Lock (Sections 4.1 to 4.3) (Section 8);
- we implement our CAC scheme in a modular and portable open-source tool — named CryptoAC.[6] Then, we conduct a thorough experimental evaluation to analyze the scalability and performance of our scheme — when implemented in CryptoAC and applied to MQTT — with respect to both a baseline configuration (i.e., with no security mechanism) and a TLS-based configuration (Section 9).
- We discuss and compare the security properties (e.g., end-to-end protection, integrity, forward secrecy, replay attack protection) of our scheme with respect to TLS (Section 10).

As a side note, we acknowledge that the use of cryptography alone to enforce AC policies makes it difficult — if possible at all — to evaluate permissions based on dynamic and contextual conditions (e.g., time-based authorizations). However, we can mitigate this limitation by combining CAC with traditional (e.g., centralized) AC enforcement mechanisms, though

[3] `https://www.kaspersky.com/blog/geolocation-data-broker-leak/53050/`

[4] `https://docs.aws.amazon.com/iot/latest/developerguide/iot-policies.html`

[5] `https://mosquitto.org/documentation/dynamic-security/`

[6] `https://github.com/stfbk/CryptoAC`

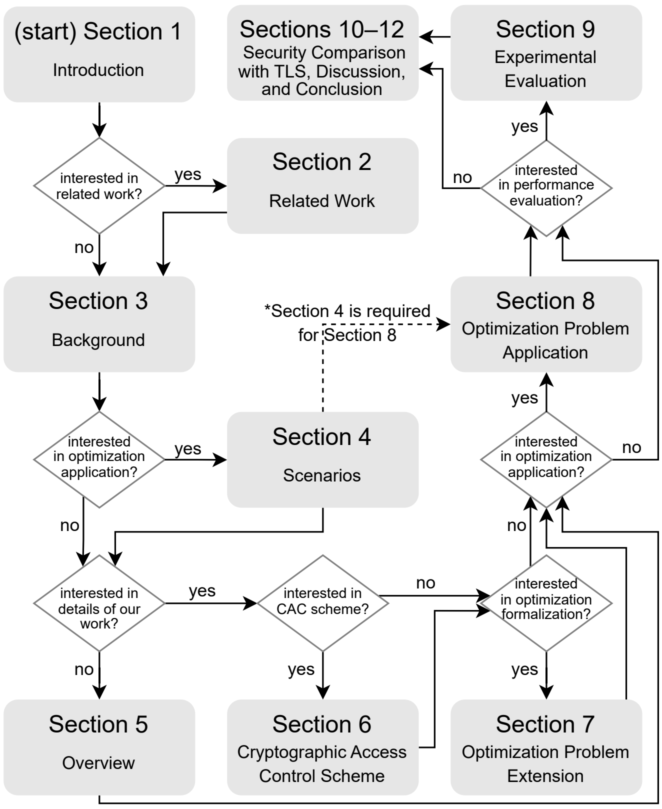

Figure 1: Paper Reading Key

this introduces the risk of potential collusion between users and the (agents managing the) enforcement mechanisms. In other words, rather than supplanting existing approaches to AC, CAC can complement and synergize with them to provide even more comprehensive protection of sensitive data.

Paper Structure. The paper is structured as follows. In Section 2, we compare our approach with related work, while in Section 3 we report the background. In Section 4, we describe and analyze the three scenarios (Remote Patient Monitoring, Cooperative Maneuvering, and Smart Lock) used as running examples throughout the paper — specifically, in Section 8. In Section 5, we give an overview of our approach. We provide a more detailed description of our CAC scheme and the optimization problem in Sections 6 and 7, respectively. We describe a proof-of-concept application of our optimization problem to the three scenarios in Section 8, and evaluate the performance and security of our CAC scheme with respect to TLS in Sections 9 and 10, respectively. We conclude the paper with a discussion on the implications and limitations of our work in Section 11 and final remarks with future work in Section 12. The reader can, of course, peruse all of the aforementioned sections to gain a thorough understanding of our work. Nonetheless, to help in navigating among the (many) concepts discussed in our paper, we provide a reading key in Figure 1 outlining different reading paths according to the reader's interests.

Conference Paper Extension. The contributions of this paper are built on top of the results we previously presented in [15], where we propose the first design of our CAC scheme and conduct a preliminary experimental evaluation — limited to measuring the performance in publishing one MQTT message at a time. This paper broadens the scope of [15] (see Table 1) by improving the design of the CAC scheme to enable (pseudo-)anonymization of the AC policy from the point of the users' perspective(Section 6.1). Additionally, we adapt the optimization problem in [12] — originally conceived for Cloud-based applications only — and extend it to IoT- and Edge-based applications (Section 7). Moreover, we provide a concrete application of the extended optimization problem to three different scenarios and discuss the obtained results (Sections 4 and 8). Finally, we expand the experimental evaluation of our CAC scheme by analyzing its scalability, assessing the performance of administrative actions, and elaborating on the comparison with TLS in terms of both performance and security (Sections 9 and 10).

2. Related Work

The topic of preserving the confidentiality and integrity of data in IoT applications has been widely addressed in the literature with a variety of approaches tackling a multitude of needs. For instance, the work in [25] introduces a blockchain-driven model for data privacy in the IoT for an eHealth scenario, emphasizing scalable data sharing under a collaborative learning paradigm. While such approaches often highlight robust decentralization for IoT applications, many do not specifically focus on decentralized AC enforcement and end-to-end protection for topic-based publish-subscribe protocols — key concerns that our work addresses. Hence, below we focus on the most relevant related work proposing security solutions for the protection of sensitive data in IoT applications explicitly employing the MQTT protocol (or equivalent protocols). During the analysis of these works, we collect key security and functional prop-

Table 1: Conference Paper Extension

	Conference Paper [15]	This work
Scenarios (Section 4)	Smart Lock	Smart Lock, Remote Patient Monitoring, Cooperative Maneuvering
CAC Scheme Design (Section 6)	Complete design	Complete design plus AC policy pseudo-anonymization
Optimization Problem (Sections 7 and 8)	—	Extension of [12] plus application to the three scenarios
CAC Scheme Evaluation (Section 9)	Preliminary performance evaluation consisting in measuring the publishing time of 1 MQTT message	Thorough performance evaluation with 4 experiments plus comparison with TLS
Security w.r.t. TLS (Section 10)	—	Analysis and comparison of security properties offered by our CAC scheme with respect to TLS

Table 2: Comparison with Related Work

	[24]	[59]	[20]	[60]	[47]	[30]	[37]	[62]	[65]	[61]	[32]	[39]	[19]	[5]	[41]	Ours
Channel encryption	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
End-to-end encryption	✗	✗	✗	✓	✗	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓	✓
Integrity guarantee	✗	✓	✗	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✓
AC policy enforcement	✓	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗	✓	✗	✗	✓	✓
Scalable w.r.t. #subscribers	✓	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓	✗	✓	✗	✓	✓
Context Awareness	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓*
Suit constrained IoT devices	✓	✓	✓	✗	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✗	✓
Different Levels of Security	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

*as mentioned in Section 1 and discussed at the end of Section 6.2, we can easily complement our CAC scheme with traditional AC enforcement mechanisms for context awareness

erties and present them in Table 2 to compare our approach with related work.

In [24] the authors design a general Attribute-Based Access Control (ABAC) enforcement mechanism for MQTT-based IoT applications. The proposed solution, which emphasizes context awareness, relies on a logically centralized entity (i.e., the reference monitor) controlling the flow of MQTT messages. While being easy to integrate into any already existing deployment, the solution requires full trust on the agent hosting the MQTT broker. Additionally, it does not guarantee the confidentiality and integrity of MQTT messages through cryptography. Similarly to our approach, the proposed solution is scalable with respect to the number of subscribers and publishers and allows enforcing AC policies. However, it does not provide integrity or end-to-end encryption.

A commonly considered solution to guarantee confidentiality and integrity in MQTT is the use of TLS. However, constrained IoT devices may struggle to support TLS. For this reason, in [59] the authors propose an alternative lightweight security mechanism to authenticate clients and secure communications. In the proposed solution, each MQTT client, as well as the MQTT broker, has a smart card, i.e., a tamper-proof hardware element with limited computational power containing secret cryptographic material. Specifically, each smart card contains a pair of asymmetric keys plus the public key of the broker. Whenever an MQTT client wants to connect to the broker, the smart card generates a random number (nonce) and encrypts it with its private key. The broker replies similarly, generating a random number to use as a symmetric key to encrypt and sign communications between the client and the broker. The use of both asymmetric and symmetric cryptography is often referred to as *hybrid cryptography*, and it is fundamental to reduce computational overhead, especially on constrained devices. While ensuring confidentiality and integrity, this solution requires the broker to decrypt all published messages and re-encrypt them for each recipient (i.e., for each subscribed MQTT client) separately, yielding a non-negligible overhead. Moreover, the confidentiality of data is preserved against Man-in-the-Middle (MitM) attackers but not from the (partially trusted provider hosting the) broker. Like our approach, the proposed solution adopts hybrid cryptography and preserves data integrity. However, it is not scalable, does not address AC policy enforcement, and lacks end-to-end encryption.

An interesting work pertaining to integrity that is surely worth citing is MQTT-I [19], a protocol extension for MQTT focused on providing not only end-to-end data flow integrity, but also completeness, correctness, and liveness of MQTT messages. The design of MQTT-I is based on Merkle trees, whose use is tailored to the context of IoT applications — i.e., decoupling between publishers and subscribers which dynamically join and leave the communication and exchange data over many topics. The authors consider the MQTT broker to be untrusted and expect the use of aggregate signatures over digests to reduce the number of signature creations (publisher-side) and verifications (subscriber-side). Intuitively, neither AC nor end-to-end encryption is the focus of the work in [19].

In [20], the authors rely on the Augmented Password-Only Authentication and Key Exchange (AugPAKE) protocol[7] to protect MQTT communications. Each MQTT client establishes a symmetric key with the broker when first connecting. Whenever a publisher creates a new topic, it derives an authorization token combining the symmetric key with the identifier of the new topic. The token is then distributed through a "secondary secure channel" to all MQTT clients the publisher authorizes to subscribe to the topic. Ultimately, the broker is responsible for verifying authorization tokens. As in [59], MQTT messages are encrypted with the publisher's symmetric key and then re-encrypted for each subscriber, making the solution hardly scalable. Moreover, the authors themselves note that the broker must be trusted and can access plaintext MQTT messages. Finally, the authors do not discuss mechanisms to provide data integrity.

To prevent MQTT brokers from accessing plaintext data, Segarra et al. [60] integrate the ARM TrustZone Trusted Execution Environment (TEE)[8]. In the proposed solution, a client securely connects to the broker using TLS. Then, the client generates a symmetric key, encrypts it with the public key of the broker-side TEE, and sends it to the TEE. From there onwards, the client encrypts all published MQTT messages using

[7] `https://datatracker.ietf.org/doc/draft-irtf-cfrg-augpake/`

[8] `https://developer.arm.com/technologies/trustzone` within the implementation of the Mosquitto MQTT broker

the symmetric key shared with the TEE, which decrypts and consequently re-encrypts messages for each subscriber. This approach avoids disclosing sensitive data to the broker (as opposed to [59, 24]), even though incurring an overhead up to 8×. Moreover, the use of TLS is not always possible, and the scalability is limited by the per-subscriber re-encryption process.

Starting from the consideration that IoT applications are often characterized by a trade-off between security and quality of service, in [47] the authors propose a framework offering 3 increasing security levels for communications in MQTT. The first level provides data integrity, authenticity, and accountability and it is designed for lightweight data exchanges. The second level adds data confidentiality and anonymity for MQTT clients. Finally, the third level offers long-term security and mutual authentication for all parties involved. Having different security levels allows for adapting the solution to the performance goals of different IoT applications (e.g., latency, scalability, computational power available), though no security level provides end-to-end encryption or considers the enforcement of AC policies as we do.

Starting from the consideration that TLS is often computationally unsustainable for IoT devices, Fan et al. [32] propose a data protection mechanism for MQTT based on the hierarchical structure of MQTT topics (see Section 3.2). In detail, the authors employ Hierarchical Identity-Based Encryption (HIBE) for encrypting each MQTT message according to what topic the message is published into. The basic intuition is that the symmetric key used to protect messages published in child topics can be derived from the symmetric key used to protect messages published in parent topics. In this way, resource-constrained IoT devices can derive multiple keys without heavy interaction with centralized authorities. At the same time, the authors' solution does not natively consider AC enforcement, as HIBE does not inherently specify policy constructs such as roles or attributes. Moreover, while providing a natural and intuitive mapping between cryptographic keys and topics structure, the authors' solution requires the MQTT broker to act as Public Key Generator (PKG). Consequently, the partially trusted agent managing the broker can eavesdrop on communications and violate data confidentiality (and, possibly, even integrity) — negating true end-to-end protection.

Similarly to [32], Akshatha and Dilip Kumar [5] propose an approach alternative to TLS providing end-to-end encryption of MQTT messages using Fernet[9] — essentially, 128-bit AES in Cipher Block Chaining (CBC) mode with the SHA-256 hash function — for IoT communications using MQTT. Since the key is only generated once, less time is needed for long-lasting communications. While providing data confidentiality and integrity, the proposed solution does not inherently incorporate AC, nor does it address dynamic policy updates or revocations.

In [30], the authors propose a modification to the MQTT protocol for providing data confidentiality and integrity in an eHealth scenario where multiple wearable devices (e.g., smartwatches, pacemakers) communicate with a single entity (e.g., a doctor). The modification focuses on three steps, namely authentication, key establishment, and data exchange. Authentication happens with username-password pairs combined with Elliptic Curve Digital Signature Algorithm (ECDSA). For key establishment, each publisher establishes a symmetric key with the (unique) subscriber through Elliptic-curve Diffie–Hellman (ECDH). Finally, data exchange happens by protecting data with the symmetric key previously established. Similarly to our approach, the solution proposed in [30] provides end-to-end encryption, integrity, and is suitable for constrained IoT devices. However, the proposed solution is applicable to IoT applications expecting many-to-1 communication only and does not consider the enforcement of AC policies — which is delegated to the MQTT broker.

Works based on Attribute-based Encryption. The authors of [37] propose an end-to-end encryption layer for MQTT based on Ciphertext-Policy Attribute-Based Encryption (ABE). In particular, the authors consider an Industry 4.0-based scenario where MQTT clients consist of, e.g., sensors and actuators, and the MQTT broker is honest-but-curious. The authors consider four steps, namely, setup, encryption, publish, and decryption: MQTT clients first obtain suitably-defined private ABE keys to then encrypt and publish MQTT messages — which are decrypted by subscribers using CP-ABE as well. Although interesting, the proposed encryption layer seems not to employ hybrid cryptography, possibly entailing non-negligible overhead.

In [62], the authors propose using ABE to secure communications in MQTT. Specifically, the authors consider both CP-ABE and Key-Policy ABE (KP-ABE). During setup, publishers and subscribers can register toward a PKG entity to obtain private keys embedding their attributes (if CP-ABE) or their access policy (if KP-ABE). Published MQTT messages are encrypted with 128-bit AES keys, which in turn are encrypted with publishers' ABE keys. Consequently, subscribers can decrypt MQTT messages only if authorized — i.e., their private ABE key embeds an authorized attribute set (if CP-ABE) or the embedded policy is satisfied (if KP-ABE). The authors highlight that CP-ABE is more flexible but incurs greater overhead than KP-ABE. On the other hand, KP-ABE requires either constant interaction between MQTT clients and the PKG or the definition of keys and a standard set of AC policies a-priori. Like our approach, the solution proposed in [62] provides end-to-end encryption, AC enforcement, and seems to scale well. 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 data integrity.

In [39], the authors aim at securing MQTT communication by employing hybrid cryptography. Specifically, they combine (a dynamic variant of) AES with KP-ABE. As intuitive, AES is used for en/decrypting MQTT messages, while KP-ABE is used to distribute symmetric keys to (authorized) MQTT clients. While proposing a straightforward and interesting solution for data protection in IoT applications, the authors do not discuss the dynamicity of the AC policy — similarly to [62] — nor the integrity of MQTT messages (although integrity could be easily achieved using a suitable mode of operation for AES

[9] `https://github.com/fernet/spec/blob/master/Spec.md`

such as in Galois/Counter Mode — GCM). Then, while providing an extensive performance evaluation, the authors do not discuss directly the scalability of their solution. Notably, the authors assume that all clients and the broker are honest but curious.

Similarly to [62, 39], also the authors in [65] combine ABE and symmetric cryptography to protect data exchanged by IoT devices through the MQTT protocol. Each MQTT client is equipped with a CP-ABE private key for symmetric key distribution. In turn, symmetric keys are used for guaranteeing end-to-end confidentiality (but not integrity) of data. The authors in [65] focus on application-level forward and backward secrecy, whereby private ABE and symmetric keys are renewed whenever an MQTT client joins or leaves a specific topic — which essentially correspond to the concept of revocation in CAC (that we discuss in Section 5.1).

In [41], the authors investigate the use of Searchable Encryption (SE) together with ABE in IoT applications employing publish-subscribe protocols. In detail, the authors propose a privacy-oriented architecture for data dissemination distributed at the Edge: publishers generate and encrypt data with ABE and SE, sharing the encrypted data with servers located at the Edge, while subscribers issues queries (e.g., keyword search) to such servers which execute queries over encrypted data. The authors assume the use of TLS for protecting all communications, but do not consider dynamic key revocation features typically needed when modifying the AC policy.

Sicari et al. [61] combine ABE with sticky policies for data protection in a smart home IoT application. In essence, sticky policies are AC policies that travel along with the associated data and are evaluated by newa trusted authority. Moreover, the authors propose a cross-domain middleware platform to relieve some of the drawbacks of both ABE and sticky policies. However, the encryption of data occurs within the platform and not in IoT devices. Although relieving the computational overhead on IoT devices, this choice prevents true end-to-end encryption.

The aforementioned works all propose interesting solutions for securing communications in MQTT-based IoT applications, each considering a specific context (e.g., eHealth, industry 4.0), technology (e.g., smart card, TEE) or cryptographic primitive (e.g., ABE). However, as shown in Table 2, these works only partially overlap with ours. In particular, end-to-end encryption and AC policies enforcement for many-to-many communications — together with quality of service optimization — has seldom been considered in the literature and is a novel contribution of our approach.

3. Background

Below, we give an overview of of AC (Section 3.1) and the MQTT protocol (Section 3.2).

3.1. Access Control

AC is "the process of mediating every request to resources maintained by a system and determining whether the request should be granted or denied" [57]; a resource usually contains sensitive data, while a request consists in asking to perform an operation over a resource (e.g., read a file). AC is traditionally divided into three levels:

- *Policy*: this abstract level comprises the rules stating which users can perform which operations on which resources. The policy is usually defined by the owner of the resources or the administrator of the system;
- *Model*: this intermediate level comprises a formal representation of the policy (e.g., RBAC [58] and ABAC [40] are two models) giving the semantics for granting or denying users' requests;
- *Enforcement*: this concrete level comprises the entities (e.g., software) that enforce the policy in the chosen model.

In what follows, we assume an AC policy $\mathbf{P}$ to be compiled into a RBAC model rather than an ABAC model (as in [24]), since support for the enforcement of RBAC policies is readily available in several broker implementations of topic-based publish-subscribe protocols, thus simplifying the experimental validation of our approach (see Section 3.2). The state of $\mathbf{P}$ can be described as a tuple $(\mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{UR}, \mathbf{PA})$, 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 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 $\exists r \in \mathbf{R} : (u, r) \in \mathbf{UR} \wedge (r, \langle f, op\rangle) \in \mathbf{PA}$; we note that role hierarchies can always be compiled away by adding suitable pairs to $\mathbf{UR}$. We report the complete set Ψ of available actions for core RBAC — according to the National Institute of Standards and Technology (NIST) standard [33] — and the resulting states in Table 3. In detail, we write $\mathbf{P} \mapsto_\psi \mathbf{P}'$ to denote that the action $\psi \in \Psi$ transforms the state $\mathbf{P}$ into the state $\mathbf{P}'$.

3.2. The MQTT Protocol

MQTT is a lightweight topic-based publish-subscribe messaging protocol[10] widely employed in IoT applications [51].

[10] `https://www.iso.org/standard/69466.html`

Table 3: Set Ψ of Core RBAC Actions According to [33]

RBAC Action ψ	Resulting Policy State
$\underline{addUser(u)}$	$\langle \mathbf{U} \cup \{u\}, \mathbf{R}, \mathbf{F}, \mathbf{UR}, \mathbf{PA}\rangle$
$\underline{deleteUser(u)}$	$\langle \mathbf{U} \setminus \{u\}, \mathbf{R}, \mathbf{F}, \mathbf{UR}, \mathbf{PA}\rangle$
$\underline{addRole(r)}$	$\langle \mathbf{U}, \mathbf{R} \cup \{r\}, \mathbf{F}, \mathbf{UR}, \mathbf{PA}\rangle$
$\underline{deleteRole(r)}$	$\langle \mathbf{U}, \mathbf{R} \setminus \{r\}, \mathbf{F}, \mathbf{UR}, \mathbf{PA}\rangle$
$\underline{addResource(f)}$	$\langle \mathbf{U}, \mathbf{R}, \mathbf{F} \cup \{f\}, \mathbf{UR}, \mathbf{PA}\rangle$
$\underline{deleteResource(f)}$	$\langle \mathbf{U}, \mathbf{R}, \mathbf{F} \setminus \{f\}, \mathbf{UR}, \mathbf{PA}\rangle$
$\underline{assignUserToRole(u, r)}$	$\langle \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{UR} \cup \{(u, r)\}, \mathbf{PA}\rangle$
$\underline{assignPermissionToRole(r, \langle f, op\rangle)}$	$\langle \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{UR}, \mathbf{PA} \cup \{(r, \langle f, op\rangle)\}\rangle$
$\underline{revokeUserFromRole(u, r)}$	$\langle \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{UR} \setminus \{(u, r)\}, \mathbf{PA}\rangle$
$\underline{revokePermissionFromRole(r, \langle f, op\rangle)}$	$\langle \mathbf{U}, \mathbf{R}, \mathbf{F}, \mathbf{UR}, \mathbf{PA} \setminus \{(r, \langle f, op\rangle)\}\rangle$

MQTT messages are 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). Each MQTT topic is identified with a case-sensitive UTF-8 encoded name. Topics are created in a hierarchy similar to folders and files in a file system using the forward-slash character "/" as a separator. *MQTT clients* (e.g., IoT devices) can subscribe to a topic, thus expressing the will to receive messages published to that topic. Whenever a 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*; the broker 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.

Topic Life-cycle. A MQTT topic is created whenever a client subscribes or publishes a message to the topic. The broker keeps the topic alive as long as either at least one client is subscribed to the topic or a client published a message asking the broker to *retain* that message. A retained message is stored by the broker and sent to each client that subscribes to the topic. A topic may have one and one only retained message, and the publishing of a new retained message overwrites the previous one. This mechanism implies that a client will not receive messages published before its subscription to a topic, except for the retained message (if present).

MQTT Implementations and Security Extensions. Among MQTT broker implementations, it is common to find extensions supporting security mechanisms such as centralized enforcement of AC policies. Mosquitto[11] is an open-source (EPL/EDL licensed) message broker maintained by the Eclipse Foundation that implements the MQTT protocol (versions 5.0, 3.1.1 and 3.1). Mosquitto provides a variety of functionalities including the DYNSEC plugin, which enforces dynamic RBAC policies. The administrator can manage the DYNSEC RBAC policy using a special topic-based Application Programming Interface (API) that allows regulating access to four operations, i.e., subscribe, publish, send, and receive messages.

4. Scenarios

We now present three widely studied scenarios for IoT applications: Remote Patient Monitoring (Section 4.1), Cooperative Maneuvering (Section 4.2), and Smart Lock (Section 4.3). We use these 3 scenarios as running examples to demonstrate the effectiveness of our optimization problem (Section 8) and evaluate the performance of our CAC scheme (Section 9). Below, we describe each scenario, discussing its trust assumptions and performance goals, eventually providing a more general definition of the problems addressed in this paper (Section 4.4).

[11] `https://mosquitto.org/`

4.1. Remote Patient Monitoring

While providing several benefits, the remote monitoring of patients [1] introduces the problem of securing medical data (e.g., Blood sugar, LDL Cholesterol) transmission and access. Below, we summarize important requirements (*R*) of the Remote Patient Monitoring (*RPM*) scenario extracted from relevant literature.

In [48], the authors highlight the importance of protecting users' medical data collected in real-time by IoT wearables. Although these data are (mainly) used for monitoring and not for diagnoses, integrity is a valued property as well as availability (*RPM-R1*). Being computationally restrained, the IoT wearables often rely on a gateway (i.e., the user's smartphone) to transmit medical data and reduce communication delay and bandwidth consumption (*RPM-R2*). Egala et al. [29] highlight the need to decentralize the storage of medical data to avoid Single Point of Failures (SPoFs) and achieve lower latency. At the same time, they claim that a hybrid architecture combining the Edge and the Cloud yields the best results in terms of robustness and redundancy (*RPM-R3*). Unfortunately, these benefits have to be balanced with the risks posed by honest but curious Cloud and Edge providers (*RPM-R4*). For instance, Domingo-Ferrer et al [28] note that even metadata (e.g., patients' name, diseases) must be hidden from partially trusted providers, as they may still leak confidential information (*RPM-R5*). Finally, the presence of disgruntled doctors is considered unlikely, especially in specialized clinics with well-paid personnel [12] (*RPM-R6*).

4.2. Cooperative Maneuvering

Smart vehicles enable a wide array of real-time Cooperative, Connected and Automated Mobility (CCAM) services assisting drivers (or autonomous vehicles) in performing complex maneuvers securely [13]. Below, we summarize important requirements (*R*) of the Cooperative Maneuvering (*CM*) scenario extracted from relevant literature.

The authors of [22] consider two prominent CCAM services envisioned in the 5G-CARMEN[12] European project, i.e., Cooperative and Automated Lane-change Maneuver (CALM) — orchestrating lane-change maneuvers in highways — and Back-Situation Awareness (BSA) — warning drivers of approaching emergency vehicles. In both CCAM services, smart vehicles dispatch information (e.g., position, direction, and speed) through roadside infrastructure nodes (*CM-R1*). In general, the authors argue that CCAM services are characterized by a delicate trade-off between performance and security. On the one hand, CCAM services must be highly scalable to support dense traffic conditions while guaranteeing ultra-low latency for up-to-date driving recommendations [54] (*CM-R2*). On the other hand, the integrity of messages is of paramount importance (*CM-R3*) to prevent potentially dramatic safety issues (e.g., due to incorrect maneuvering recommendations). Moreover, communication channels are considered at high risk due to the public nature of the (wireless) network [13] (*CM-R4*). Availability is not as important, as the safety impact for an unavailable

[12] `https://5gcarmen.fbk.eu/`

CCAM service is often mitigated by designing fallback mechanisms to avoid worst-case scenarios [22] (*CM-R5*). For instance, when the CALM service is unavailable, the vehicles themselves could issue a warning to their drivers and provide the safest advice based on the current context (e.g., slow down, do not merge). Finally, as discussed in [13], access to (smart) vehicles is protected by both high-security standards (i.e., smart keys, anti-thief security systems) and credentials (e.g., PIN, biometrics, passwords), reducing the likelihood of a physical attack (*CM-R6*).

4.3. Smart Lock

Organizations operating in large buildings (e.g., government structures, hospitals, research centers) often use smart locks[13] to regulate access to rooms, closets, and laboratories containing delicate equipment or health hazards [3, 2, 4]. Below, we summarize important requirements (*R*) of the Smart Lock (*SL*) scenario extracted from relevant literature.

As discussed in [4], a smart lock service must guarantee reliability and DoS resilience to serve incoming requests (*SL-R1*). Low latency is desirable to provide a seamless user experience but not crucial (*SL-R2*). Moreover, as the numbers of employees in an organization and smart locks in a building tend to be limited and static, scalability is not a prominent performance goal (*SL-R3*). The same applies to memory, bandwidth, and computational power, as messages exchanged with smart locks are limited to a few bytes [4] (*SL-R4*). Finally, smart-lock services must require minimal maintenance to still be preferable over traditional locks (*SL-R5*).

4.4. Problems Characterization

The scenarios presented in Sections 4.1 to 4.3 all require regulating access to valuable data (i.e., medical data, vehicles' information, smart lock tokens) while guaranteeing confidentiality and/or integrity (e.g., to prevent an attacker from tampering with driving recommendations in the CALM service). When a fully trusted central agent is missing, CAC is the natural solution to enforce AC policies while providing a high degree of protection over data [12]. However, each scenario has specific trust assumptions and performance goals (e.g., the Remote Patient Monitoring scenario is more concerned about data confidentiality, while the Smart Lock scenario prioritizes DoS resilience and availability). Moreover, assumptions and goals within the same scenario may conflict with each other (e.g., the CALM service requires low latency and, at the same time, strong integrity guarantees through robust — and computationally expensive — cryptographic primitives), leading to a stalemate in which administrators need to carefully evaluate the benefits and drawbacks of different CAC schemes architectures.

From these considerations, we can characterize the 2 problems we address in our work as follows. The first problem lies in the design of a CAC scheme for the end-to-end protection of sensitive data exchanged in IoT applications through the cryptographic enforcement of AC policies. The second problem lies in how to select the CAC scheme architecture which simultaneously minimizes the risks to the confidentiality, integrity, and availability of data — derived from the presence of certain attackers described in the trust assumptions — and maximizes the satisfaction of the goals relevant to a given scenario.

[13]A smart lock is a cyber-physical device composed of a smart cylinder and a microcontroller that requires a (digital) *token* to be unlocked. Usually, tokens are distributed to (authorized) users according to their qualifications.

5. Overview

We now give a high-level overview of our solution (see Figure 2) to the two problems discussed in Section 4.4. First, we illustrate the intuition behind the design of our CAC scheme (Section 5.1). Then, we formalize an optimization problem allowing for performing trade-off analyses of (potentially conflicting) trust assumptions and performance goals (Section 5.2). The objective of this section is to provide the reader with a general understanding of our solution before delving into (complex) details; we provide a more meticulous description of the design of the CAC scheme and the optimization problem in Sections 6 and 7, respectively.

5.1. Cryptographic Access Control Scheme Design

CAC enforces AC policies while guaranteeing the confidentiality and the integrity of sensitive data through cryptography. For this reason, CAC is suitable in scenarios lacking a fully trusted central agent. Usually, CAC assigns a unique symmetric key to each resource (e.g., a document, an MQTT topic) used to en/decrypt the (sensitive data contained in) the resource; resources are entrusted to an entity called Data Manager (DM) (e.g., an MQTT broker) [12]. Then, each user is equipped with — and assumed to be correctly associated with — a pair of asymmetric cryptographic public-private keys. To assign a user the permission to read a resource, the symmetric key of the resource is encrypted with the public key of the user. In this way, the user can obtain the symmetric key using his private key and decrypt the sensitive data contained in the resource. All cryptographic computations are performed by (one or more instances of) an entity called *proxy*, which typically operates at the client side (e.g., users' laptops, smartphones, or IoT devices).

CAC Policy Encoding. In our context (i.e., topic-based publish-subscribe protocols in IoT applications) we consider a RBAC policy (see Section 3.1): the clients of the publish-subscribe protocol are the users, while the topics are the resources. Roles, instead, are defined by the administrator(s) according to, e.g., the internal hierarchy of their organization or the hierarchical structure of the topics. Similarly to users, each role is associated with a pair of asymmetric cryptographic public-private keys; this introduces a *level of indirection* in the CAC scheme. In other words, the administrator encrypts the symmetric key associated with a resource using the public keys of those roles to which the administrator wants to grant permission to access the resource. Consequently, the administrator encrypts a role's private key using the public keys of those users the administrator wants to assign to that role. We highlight that the management

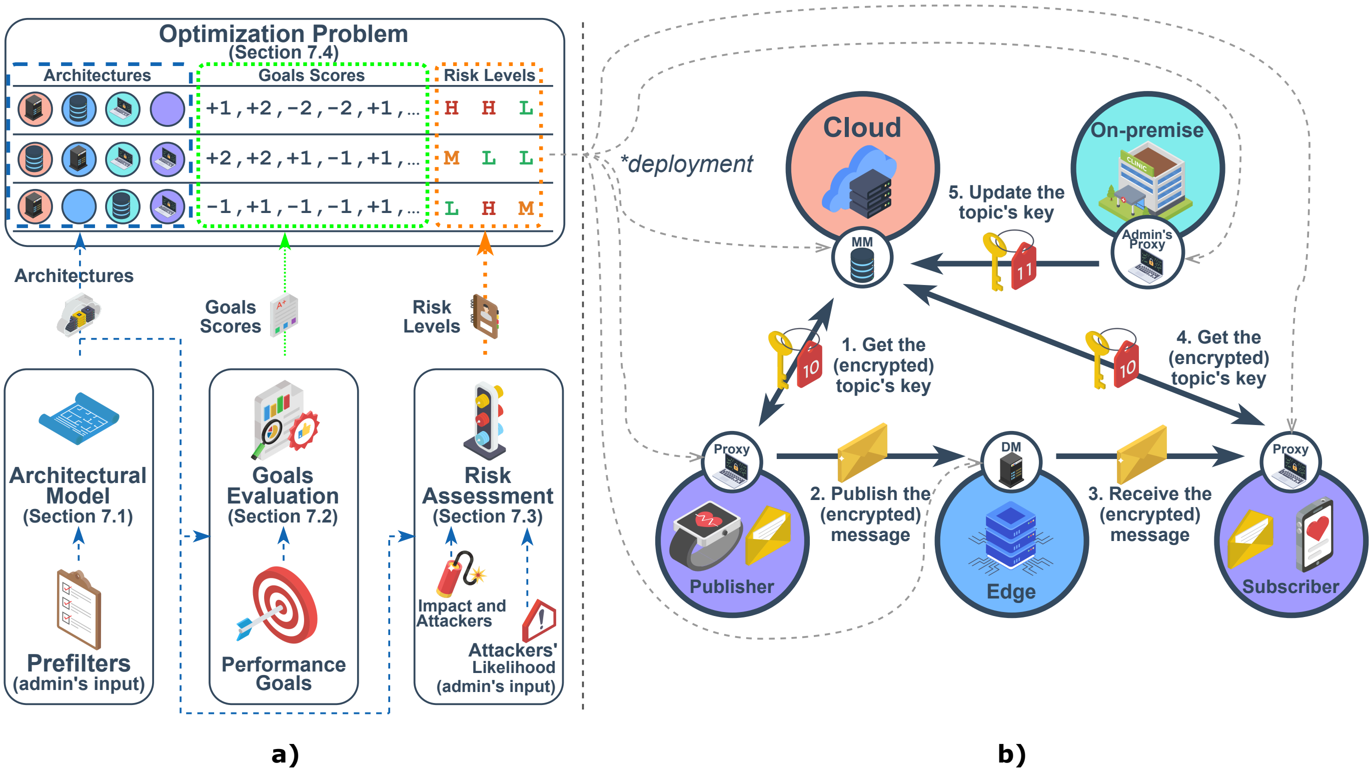

Figure 2: Approach Overview

of the AC policy is still centralized — avoiding synchronization and consistency issues — while the enforcement is precompiled and embedded into the (distribution of the) cryptographic keys — solving the problems usually affecting centralized solutions (see Section 1).

CAC Policy Functioning. A user (i.e., the publisher) can obtain a resource's symmetric key (step 1 in Figure 2b) for — encrypting and then — publishing a message to that resource (step 2) by (*i*) using her private key to get the private key of an assigned authorized role and then (*ii*) using the role's private key to get the resource's symmetric key. The encrypted message is then delivered to the DM (i.e., the broker); being protected with cryptography, tampering attempts would be obvious, while neither the (agent operating the) DM nor eventual external attackers can read the message. Then, the DM broadcasts the encrypted message (step 3) to other users (i.e., the subscribers) which — if authorized — can obtain and use the resource's symmetric key (step 4) to decrypt the encrypted message.

CAC Policy Revocation. Whenever a user is revoked access to a resource (i.e., whenever a permission is revoked), any resource's symmetric key and role's private key that the revoked user could access and — after revocation — lost access to, need to be renewed and re-distributed (step 5). In fact, the revoked user could still access the resource using a cached key. Similarly, if a symmetric or private key gets compromised (e.g., stolen by an external attacker), we follow the same renewal procedure. To keep track of the changes (and prevent replay attacks), resources' symmetric keys and roles' private keys are assigned a version number; intuitively, the version number increases by 1 every time the associated key is rotated. The version numbers — along with the state of the AC policy — compose the *metadata* which are entrusted to an entity called Metadata Manager (MM) (e.g., a Redis datastore) [12]. Metadata are digitally signed (e.g., by the administrator of the policy) to guarantee integrity and authenticity. Besides the DM, the MM, and the proxy, some CAC schemes may include a fourth entity called Reference Monitor (RM) [12]. In brief, the RM mediates users' requests for (*i*) adding new resources and (*ii*) overwriting existing resources, ensuring that users have indeed the proper permissions and that the requests are consistent. Note that the RM is never entrusted with the resources themselves, which are managed by the DM. In our context, the RM is not needed as (*i*) only the administrator can add resources and (*ii*) users cannot overwrite previously sent MQTT messages (i.e., writing data to a resource is equivalent to appending data to the resource).

Contextualization with Respect to Other Security Solutions. CAC is just one part of the overall security of IoT applications. In particular CAC addresses 2 security aspects: authorization (i.e., enforcement of AC policies) and secure communication for end-to-end data protection (through cryptography).

In the context of authorization, a security solution possibly related to CAC is OAuth 2.0[14]. However, we note that OAuth 2.0 is a standard for authorization delegation of third parties,

[14] `https://oauth.net/2/`

not for enforcing AC policies (nor encrypting data). Then, when seen as an AC enforcement mechanism, CAC may be compared to traditional mechanisms such as the DYNSEC plugin for Mosquitto (see Section 3.2 but also the eXtensible Access Control Markup Language (XACML) standard[15] and the Open Policy Agent (OPA)[16]. A clear difference between CAC and these other mechanisms is that CAC enforces AC policies cryptographically, while these other mechanisms enforce AC policies in a centralized manner without applying cryptography over data.

In the context of secure communication for data protection, CAC can be compared with TLS; however, as we discuss at length in Section 10, CAC and TLS have noticeable differences. Then, network-level security solutions such as network segmentation, firewalls, intrusion detection and prevention systems, and network security protocols (e.g., wireless security protocols such as WPA3 — particularly relevant for IoT devices) simply operate at a different level with respect to CryptoAC in the ISO/OSI and TCP/IP stacks.[17]

Another relevant security aspect is authentication, for which popular standards are OpenID Connect[18] and FIDO2[19]. Authentication is often a prerequisite of authorization; still, these two security aspects are clearly distinct.

In conclusion, CAC is an optimal solution for combining authorization and secure end-to-end communication. However, it is important to highlight that CAC should be regarded as complementary to the aforementioned security solutions rather than as a replacement. In other words, a truly comprehensive and robust security strategy requires integrating CAC with the aforementioned (and also further) security solutions.

5.2. Goals Satisfaction and Risks Minimization

Once defined (the functions and the interactions among) the entities of the CAC scheme (see Section 5.1), we need to decide where to deploy them. In Cloud-Edge-IoT applications, we identify 4 distinct logical/physical locations — which we call *domains* — that can host entities: client, on-premise (i.e., within the organization), Edge, and Cloud. The assignment of entities to domains (see Figure 2b) comprises the *architecture* of the CAC scheme, while the *architectural model* comprises all possible CAC schemes architectures (i.e., all possible assignments of entities to domains). Of course, the administrator can exclude certain architectures by specifying some *prefilters*. As we argue below, the choice of where — i.e., in which domain(s) — to deploy a certain entity impacts both the trust assumptions and the performance goals relevant to the underlying scenario.

Goals Evaluation. The architecture of a CAC scheme may influence the achievement of performance goals such as low latency, high bandwidth, and scalability. For instance, assigning the DM (i.e., the broker) to the Cloud domain may favor reliability and benefit from higher computational power — at the cost of increasing latency with respect to the Edge domain. Generalizing, we can determine how much an architecture attains a certain goal by calculating a goal score (i.e., an integer number), whose value depends on how each entity-domain assignment of the architecture affects (e.g., positively or negatively) that goal. By replicating this approach for all goals, we obtain an array of *goals scores* (one score for each goal) for each architecture.

Risk Assessment. We can assume that CAC schemes do not have any vulnerability or design error — and that cryptographic primitives cannot be broken. Nonetheless, there still are further threats and attackers to consider. For instance, Zhou et al. [69] identify a list of potential attackers in outsourced healthcare storage systems. First, physical devices (e.g., IoT wearables) can be targeted by unauthorized users and Man-at-the-Device (MatD) attackers, especially if not attended or during (the confusion arising in the course of) a medical emergency. Second, partially trusted Edge and Cloud providers may collude with malicious insiders (e.g., disgruntled employees) to bypass security measures (e.g., smart locks). Thirdly, MitM attackers may threaten the security of communication channels. According to the trust assumptions of their scenario, administrators can specify a *likelihood* on the presence of each of these attackers. In this way, it is possible to evaluate — for each CAC scheme architecture — the corresponding *risk levels* through a *risk assessment methodology* combining the given likelihood with a (precomputed) impact. Intuitively, if an administrator regards the presence of disgruntled employees as likely, an architecture expecting to deploy all entities within the organization premises would have a higher risk level than an architecture deploying entities in the Edge or the Cloud.

Optimization Problem. Once defined the risk levels and the goals scores for each CAC scheme architecture, we can formalize an optimization problem to find the architecture which minimizes the former and maximizes the latter. One possible solution consists in identifying the set of *Pareto Optimal* architectures [44] — that is, architectures which can be considered equally good. An architecture is Pareto Optimal if there does not exist another architecture that improves one goal score or reduces a risk level without detriment to another. Another possible solution is to *compress* the goals scores and the risk levels into a single value (e.g., by calculating a weighted sum) for each architecture, and then rank all architectures — according to this value — and pick the best (i.e., the first in such a ranking).

6. Cryptographic Access Control Scheme

We now present a role-based CAC scheme for the end-to-end protection of sensitive data exchanged through topic-based publish-subscribe protocols in IoT applications. Our CAC considers MQTT — which is the de-facto standard for communication in IoT [51] — but can easily adapt to similar protocols (e.g., AMQP[20]). The design of our scheme is inspired by the

[15] https://www.oasis-open.org/committees/xacml/
[16] https://www.openpolicyagent.org/
[17] https://www.ibm.com/think/topics/osi-model
[18] https://openid.net/
[19] https://fidoalliance.org/fido2/
[20] https://www.amqp.org/

work in [34] with several technical variations — discussed below — and two notable differences, namely the context of use (i.e., Cloud in [34] vs. IoT in this work) and the protection of data (i.e., at rest in [34] vs. in transit in this work). Below, we first show how to map RBAC elements (e.g., users, roles, resources) to MQTT clients and MQTT topics. (Section 6.1). Then, we present the full construction and pseudocode of our CAC scheme (Section 6.2) and informally discuss its security (Section 6.3). Finally, we describe the implementation of our scheme (Section 6.4). We report the symbols used throughout this section in Table 4.

Table 4: Symbols

Symbol	Description
e	An element (either a user, a role or a resource)
u	A user
A	The administrator user
r	A role
f	A resource
v_e	A version number for the element e
$p_{(e,v_e)}$	A pseudonym for e with version number v_e
op	Either $\{\mathsf{Sub}\}$, $\{\mathsf{Pub}\}$ or $\{\mathsf{Sub}, \mathsf{Pub}\}$
Enf	Either $\{\mathsf{Enf_c}\}$, $\{\mathsf{Enf_t}\}$ or $\{\mathsf{Enf_c}, \mathsf{Enf_t}\}$
$\mathbf{N}$	Null (i.e., empty) value
fc	The content of a message
c	A ciphertext
$-$	Wildcard
$\mathbf{Gen^{Pub}}$	Generation of key pair for en/decryption
$\mathbf{Gen^{Sig}}$	Generation of key pair for signatures
$\mathbf{Gen^{Sym}}$	Generation of symmetric key
$\mathbf{Gen^{Pse}}$	Generation of pseudonym
$\mathbf{k}^{\mathbf{enc}}_{(e,v_e)}$	Public encryption key of e with version number v_e
$\mathbf{k}^{\mathbf{dec}}_{(e,v_e)}$	Private decryption key of e with version number v_e
$\mathbf{k}^{\mathbf{ver}}_{(e,v_e)}$	Public verification key of e with version number v_e
$\mathbf{k}^{\mathbf{sig}}_{(e,v_e)}$	Private signing key of e with version number v_e
$\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$	Symmetric key of f with version number v_f
$\mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(e,v_e)}}(-)$	Encryption of $-$ with key $\mathbf{k}^{\mathbf{enc}}_{(e,v_e)}$
$\mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_{(e,v_e)}}(-)$	Decryption of $-$ with key $\mathbf{k}^{\mathbf{dec}}_{(e,v_e)}$
$\mathbf{Ver}^{\mathbf{Sig}}_{\mathbf{k}^{\mathbf{ver}}_{(e,v_e)}}(-)$	Creation of signature for $-$ with key $\mathbf{k}^{\mathbf{ver}}_{(e,v_e)}$
$\mathbf{Sign}^{\mathbf{Sig}}_{\mathbf{k}^{\mathbf{ver}}_{(e,v_e)}}(-,-')$	Verification of signature $-$ for $-'$ with key $\mathbf{k}^{\mathbf{sig}}_{(e,v_e)}$
$\mathbf{Enc}^{\mathbf{S}}_{\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}}(fc)$	Symmetric of $-$ encryption with key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$
$\mathbf{Dec}^{\mathbf{S}}_{\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}}(c)$	Symmetric of $-$ decryption with key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$
$\langle \mathbf{U_t}, \mathbf{R_t}, \mathbf{F_t}, \mathbf{UR_t}, \mathbf{PA_t} \rangle$	The state of the AC policy enforced traditionally ($\mathbf{P_t}$)
$\mathbf{U_t}$	Set of users; a member is a single value u
$\mathbf{R_t}$	Set of roles; a member is a single value r
$\mathbf{F_t}$	Set of resources; a member is a single value f
$\mathbf{UR_t}$	Set of user-role pairs; a member is a tuple (u, r)
$\mathbf{PA_t}$	Set of role-permission pairs; a member is a tuple $(r, \langle f, op \rangle)$
$\langle \mathbf{U_c}, \mathbf{R_c}, \mathbf{F_c}, \mathbf{UR_c}, \mathbf{PA_c} \rangle$	The state of the CAC policy ($\mathbf{P_c}$)
$\mathbf{U_c}$	Set of users; a member is a tuple $(u, p_u, \mathbf{k}^{\mathbf{enc}}_u, \mathbf{k}^{\mathbf{ver}}_u)$
$\mathbf{R_c}$	Set of roles; a member is a tuple $(r, p_{(r,v_r)}, \mathbf{k}^{\mathbf{enc}}_{(r,v_r)}, \mathbf{k}^{\mathbf{ver}}_{(r,v_r)}, v_r)$
$\mathbf{F_c}$	Set of resources; a member is a tuple $(f, p_{(f,v_f)}, v_f, Enf)$
$\mathbf{UR_c}$	Set of user-role pairs; a member is a tuple $\left(u, r, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_u}\left(\mathbf{k}^{\mathbf{enc}}_{(r,v_r)} \mathbf{k}^{\mathbf{dec}}_{(r,v_r)}, \mathbf{k}^{\mathbf{ver}}_{(r,v_r)}, \mathbf{k}^{\mathbf{sig}}_{(r,v_r)}\right), v_r\right)$
$\mathbf{PA_c}$	Set of role-permission pairs; a member is a tuple $\left(r, f, p_{(r,v_r)}, p_{(f,v_f)}, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r)}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}\right), v_f, v_r, op\right)$

6.1. RBAC to MQTT

We map MQTT clients and MQTT topics to the set of users $\mathbf{U}$ and resources $\mathbf{F}$ of the RBAC policy $\mathbf{P}$, respectively. As described in Section 5.1, the administrator defines the set of roles $\mathbf{R}$ — which are not mapped to any MQTT concept. The set of operations $\mathbf{OP}$ comprises publish (Pub) and subscribe (Sub), i.e., $\mathbf{OP} = \{\mathsf{Pub}, \mathsf{Sub}\}$. Each user u and role r with version number v_r (recall the use of version numbers described in Section 5.1) is provided with two pairs of asymmetric keys $(\mathbf{k}^{\mathbf{enc}}_{u/(r,v_r)}, \mathbf{k}^{\mathbf{dec}}_{u/(r,v_r)})$ and $(\mathbf{k}^{\mathbf{ver}}_{u/(r,v_r)}, \mathbf{k}^{\mathbf{sig}}_{u/(r,v_r)})$ for en/decryption and verification/creation of digital signatures, respectively. Each resource (i.e., topic) f with version number v_f is assigned to a symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$. Intuitively, $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$ is used to en/decrypt the content fc of each message published to f, resulting in $\mathbf{Enc}^{\mathbf{S}}_{\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}}(fc)$. To create a resource f, the administrator generates a new symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$ and publishes a retained message to a new topic named f containing the version number v_f (at first, $v_f = 1$). To give a role r access to the messages published in f, f's symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$ is encrypted with r's encryption public key $\mathbf{k}^{\mathbf{enc}}_{(r,v_r)}$, resulting in $\mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r)}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}\right)$. To assign a user u to a role r, r's decryption and signature creation keys $(\mathbf{k}^{\mathbf{dec}}_{(r,v_r)}, \mathbf{k}^{\mathbf{sig}}_{(r,v_r)})$ are encrypted with u's encryption public key $\mathbf{k}^{\mathbf{enc}}_u$, resulting in $\mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_u}\left(\mathbf{k}^{\mathbf{dec}}_{(r,v_r)}, \mathbf{k}^{\mathbf{sig}}_{(r,v_r)}\right)$. To read the content fc of a message published to f, a user u:

1. u decrypts $\mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_u}\left(\mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_u}\left(\mathbf{k}^{\mathbf{dec}}_{(r,v_r)}, -\right)\right)$, obtaining $\mathbf{k}^{\mathbf{dec}}_{(r,v_r)}$;
2. u decrypts $\mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_{(r,v_r)}}\left(\mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r)}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}\right)\right)$, obtaining $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$;
3. u decrypts $\mathbf{Dec}^{\mathbf{S}}_{\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}}\left(\mathbf{Enc}^{\mathbf{S}}_{\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}}(fc)\right)$, obtaining fc.

Whenever the key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$ needs to be updated (due to, e.g., a user's revocation), the administrator generates a new symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f+1)}$. Then, the administrator replaces the retained message in f with a new one containing the (updated) version number $v_f + 1$; in this way, users — both those currently subscribed and those who will subscribe in the future — are notified of the key renewal. The new key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f+1)}$ is encrypted for authorized roles as explained at the beginning of this section. To delete a resource, the administrator removes the corresponding retained message, unsubscribes all users, and deletes the related metadata. Finally, users, roles, and resources are given a random pseudonym p to pseudo-anonymize the policy. Hence, a resource can be referred to by its pseudonym besides by its identifier. Similarly, public keys are linked with users' and roles' pseudonyms to avoid disclosing their identity. Users can use their pseudonyms when, e.g., sending messages or creating digital signatures (if required). A periodical renewal of pseudonyms based on the strategy discussed in [53] allows for maintaining their effectiveness.

Traditional and Cryptographic Enforcement. Not every resource may be worth the overhead of CAC according to, e.g., the sensitivity of its messages. As such, we consider the possibility of seamlessly integrating traditional (i.e., centrally enforced) AC within our CAC scheme. Specifically, we allow the administrator to declare which kind of enforcement *Enf* to apply over a resource f, i.e., cryptographic ($\mathsf{Enf_c}$), traditional ($\mathsf{Enf_t}$), or even both. To do so, we logically split the RBAC policy state into two separate states, i.e., $\langle \mathbf{U_t}, \mathbf{R_t}, \mathbf{F_t}, \mathbf{UR_t}, \mathbf{PA_t} \rangle$ for traditional enforcement ($\mathbf{P_t}$) — where the subscript **t** stands for traditional — and $\langle \mathbf{U_c}, \mathbf{R_c}, \mathbf{F_c}, \mathbf{UR_c}, \mathbf{PA_c} \rangle$ for cryptographic enforcement ($\mathbf{P_c}$) — where the subscript **c** stands for cryptographic. The two states are always synchronized: every modification applied to the state of a policy is mirrored into the other (we provide more details on this in Section 6.2). For instance, adding a user in $\mathbf{P_c}$ implies adding a user in $\mathbf{P_t}$ as well, although the two actions are implemented differently. By choosing each time the more appropriate enforcement for a resource, it is possible to significantly reduce the cryptographic overhead.

6.2. Full Construction

We report in Figures 3 and 4 the pseudocode of our CAC scheme. In particular, we provide the pseudocode for all core RBAC actions in Ψ reported in Table 3, to which we add:

- $\underline{init_u()}$ and $\underline{init_A()}$, generating cryptographic keys and pseudonyms for users and the administrator, respectively;
- $\underline{readResource_u(f)}$ and $\underline{writeResource_u(f, fc)}$, for allowing users to read and write resources. In our context (topic-based publish-subscribe protocols), these actions correspond to publishing messages to topics and subscribing to topics, respectively.

The integrity of the state of the AC policies $\mathbf{P_c}$ and $\mathbf{P_t}$ is guaranteed by using digital signatures, while the integrity of data is guaranteed by using Authenticated Encryption with Associated Data (AEAD) [10]; for the sake of simplicity, in Figures 3 and 4 we omit these details, as well as other trivial checks like the uniqueness of identifiers and pseudonyms. Finally, we note that our CAC scheme does not prescribe the use of specific cryptographic algorithms — the CAC scheme employs a generic AEAD symmetric algorithm for en/decryption of resources, a generic asymmetric public-private cryptographic algorithm for en/decryption of keys, and a generic digital signature cryptographic algorithm; the actual choice of cryptographic algorithms is discussed in Section 6.4.

Consistency Between Cryptographic and Traditional Policy States. The relationship between $\mathbf{P_c}$ and $\mathbf{P_t}$ can be expressed as a set of invariants on the actions in Figures 3 and 4. Formally, we define a *predicate* as a Boolean-valued function on the set of states of $\mathbf{P_c}$ or $\mathbf{P_t}$. The value of a predicate Q on a generic state $\mathbf{P}$ is denoted with $\mathbf{P} \models Q$. An action ψ leaves a predicate Q invariant iff (i.e., if and only if) $\mathbf{P} \models Q$ implies $\mathbf{P}' \models Q$ whenever $\mathbf{P} \mapsto_\psi \mathbf{P}'$ (recall the notation from Section 3.1). In detail, any action ψ in Figures 3 and 4 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 permission $\langle f, op \rangle$ iff $\exists r \in \mathbf{R_t} : (u, r) \in \mathbf{UR_t} \wedge (r, \langle f, op \rangle) \in \mathbf{PA_t}$, 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 permission $\langle f, op \rangle$ iff $\exists (r, -, \mathbf{k}^{\mathbf{enc}}_{(r,v_r)}, -, -) \in \mathbf{R_c} : (u, r, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_u}\left(\mathbf{k}^{\mathbf{dec}}_{(r,v_r)}, -, -, -\right), v_r) \in \mathbf{UR_c}$ and $(r, f, -, -\mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r)}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}\right), v_f, v_r, op) \in \mathbf{PA_c}$. In turn, by recalling from Figure 3 the actions $\underline{assignUserToRole_A(u, r)}$ and $\underline{assignPermissionToRole_A(r, \langle f, op \rangle)}$, this implies that the user u can access $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$ by using her private key $\mathbf{k}^{\mathbf{dec}}_u$ to decrypt $\mathbf{k}^{\mathbf{dec}}_{(r,v_r)}$, and then using $\mathbf{k}^{\mathbf{dec}}_{(r,v_r)}$ to decrypt $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}$, as explained at the beginning of Section 6.1. In summary, the invariants 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 at the end of Section 1, that can be checked only with a traditional enforcement mechanism.

6.3. Security Considerations

Below, we provide security considerations on critical aspects of our CAC scheme: AC policy enforcement and collusions, accountability, cryptography, key distribution, and revocation.

On Traditional Policy Enforcement and Collusions. Our CAC scheme allows administrators to enforce RBAC policies both traditionally and cryptographically. Ideally, traditional enforcement allows for refining further the permissions of the users (e.g., to specify whether a user can subscribe, publish, or perform both actions). However, it has to be noted that users could potentially collude with the (partially trusted agent operating the) traditional enforcement to bypass it and gain publish and/or subscribe privileges. Nonetheless, we highlight that this kind of collusions may happen regardless of whether traditional enforcement is used. Moreover, colluding users should have the symmetric key of a resource anyway to read or write on that resource (i.e., $\mathbf{P_c}$ should already grant publish or subscribe permissions to the colluding users on that resource). Intuitively, the same could happen if symmetric keys were stolen or leaked from IoT devices. However, the security of IoT devices themselves (e.g., concerning physical attackers or firmware vulnerabilities) is out of the scope of this paper; for a detailed analysis of this subject, we refer the interested reader to [49].

On Accountability. In our CAC scheme, accountability — which we define as the ability to map messages to the users that published them — is currently not ensured cryptographically. In fact, messages are protected with symmetric keys

$\underline{init_A()}$

- Generate encryption key pair $(\mathbf{k}^{\mathbf{enc}}_A, \mathbf{k}^{\mathbf{dec}}_A) \leftarrow \mathbf{Gen}^{\mathbf{Pub}}$ and signature key pair $(\mathbf{k}^{\mathbf{ver}}_A, \mathbf{k}^{\mathbf{sig}}_A) \leftarrow \mathbf{Gen}^{\mathbf{Sig}}$
- Add $(A, A, \mathbf{k}^{\mathbf{enc}}_A, \mathbf{k}^{\mathbf{ver}}_A)$ to $\mathbf{U_c}$, $(A, A, \mathbf{k}^{\mathbf{enc}}_A, \mathbf{k}^{\mathbf{ver}}_A, 1)$ to $\mathbf{R_c}$ and $(A, A, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_A}\left(\mathbf{k}^{\mathbf{enc}}_A, \mathbf{k}^{\mathbf{dec}}_A, \mathbf{k}^{\mathbf{ver}}_A, \mathbf{k}^{\mathbf{sig}}_A\right), 1)$ to $\mathbf{UR_c}$
- Add A to $\mathbf{U_t}$, A to $\mathbf{R_t}$ and (A, A) to $\mathbf{UR_t}$

$\underline{addUser_A(u)}$

- Add $(u, \mathbf{N}, \mathbf{N}, \mathbf{N})$ to $\mathbf{U_c}$
- Add u to $\mathbf{U_t}$

$\underline{deleteUser_A(u)}$

- $\forall (r, -, -, -, -) \in \mathbf{R_c}$ *s.t.* $(u, r, -, -) \in \mathbf{UR_c}$:
  * Invoke $\underline{revokeUserFromRole_A(u, r)}$
- Delete $(u, -, -, -)$ from $\mathbf{U_c}$
- Delete u from $\mathbf{U_t}$ and $(u, -)$ from $\mathbf{UR_t}$

$\underline{addRole_A(r)}$

- Generate encryption key pair $(\mathbf{k}^{\mathbf{enc}}_{(r,1)}, \mathbf{k}^{\mathbf{dec}}_{(r,1)}) \leftarrow \mathbf{Gen}^{\mathbf{Pub}}$, signature key pair $(\mathbf{k}^{\mathbf{ver}}_{(r,1)}, \mathbf{k}^{\mathbf{sig}}_{(r,1)}) \leftarrow \mathbf{Gen}^{\mathbf{Sig}}$ and pseudonym $p_{(r,1)} \leftarrow \mathbf{Gen}^{\mathbf{Pse}}$
- Add $(r, p_{(r,1)}, \mathbf{k}^{\mathbf{enc}}_{(r,1)}, \mathbf{k}^{\mathbf{ver}}_{(r,1)}, 1)$ to $\mathbf{R_c}$ and $(A, r, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_A}\left(\mathbf{k}^{\mathbf{enc}}_{(r,1)}, \mathbf{k}^{\mathbf{dec}}_{(r,1)}, \mathbf{k}^{\mathbf{ver}}_{(r,1)}, \mathbf{k}^{\mathbf{sig}}_{(r,1)}\right), 1)$ to $\mathbf{UR_c}$
- Add r to $\mathbf{R_t}$ and (A, r) to $\mathbf{UR_t}$

$\underline{deleteRole_A(r)}$

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

$\underline{addResource_A(f, fc, Enf)}$

- If $\mathsf{Enf_c} \in Enf$:
  * Generate symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,1)} \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$
  * Set $c = \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_A}\left(\mathbf{k}^{\mathbf{sym}}_{(f,1)}\right)$
  * Set $fc_{enc} = \mathbf{Enc}^{\mathbf{S}}_{\mathbf{k}^{\mathbf{sym}}_{(f,1)}}(fc)$
- Else:
  * Set $\mathbf{k}^{\mathbf{sym}}_{(f,1)} = \mathbf{N}$
  * Set $c = \mathbf{N}$
  * Set $fc_{enc} = fc$
- Generate pseudonym $p_{(f,1)} \leftarrow \mathbf{Gen}^{\mathbf{Pse}}$
- Add $(f, p_{(f,1)}, 1, Enf)$ to $\mathbf{F_c}$ and $(A, f, A, p_{(f,1)}, c, 1, 1, \mathbf{OP})$ to $\mathbf{PA_c}$
- Add f to $\mathbf{F_t}$ and $(A, \langle f, \mathbf{OP} \rangle)$ to $\mathbf{PA_t}$
- Publish $(f, p_{(f,1)} 1, Enf)$ as a retained message and fc_{enc} as a message to f

$\underline{deleteResource_A(f)}$

- Delete $(f, -, -, -)$ from $\mathbf{F_c}$ and $(-, f, -, -, -, -, -, -)$ from $\mathbf{PA_c}$
- Delete f from $\mathbf{F_t}$ and $(-, \langle f, - \rangle)$ from $\mathbf{PA_t}$
- Delete retained message and unsubscribe all users from f

$\underline{assignUserToRole_A(u, r)}$

- Find $(A, r, c, v_{(r,v_r)}) \in \mathbf{UR_c}$
- Decrypt $m = \mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_A}(c)$
- 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 $\mathbf{UR_t}$

$\underline{assignPermissionToRole_A(r, \langle f, op \rangle)}$

- 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 $(A, f, p_A, p_{(f,v_f)}, c_A, v_f, 1, -) \in \mathbf{PA_c}$, $(r, -, \mathbf{k}^{\mathbf{enc}}_{(r,v_r)}, -, v_r) \in \mathbf{R_c}$ and $(f, p_{(f,v_f)}, v_f, Enf) \in \mathbf{F_c}$
  * If $\mathsf{Enf_c} \in Enf$:
    · Set $c_r = \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r)}}(\mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_A}(c_A))$
  * Else:
    · Set $c_r = \mathbf{N}$
  * Add $(r, f, p_{(r,v_r)}, p_{(f,v_f)}, c_r, v_f, v_r, op)$ to $\mathbf{PA_c}$
  * Add $(r, \langle f, op \rangle)$ to $\mathbf{PA_t}$

$\underline{revokeUserFromRole_A(u, r)}$

- Find $(r, -, -, -, v_r) \in \mathbf{R_c}$
- Generate new encryption key pair $(\mathbf{k}^{\mathbf{enc}}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{dec}}_{(r,v_r+1)}) \leftarrow \mathbf{Gen}^{\mathbf{Pub}}$, signature key pair $(\mathbf{k}^{\mathbf{ver}}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{sig}}_{(r,v_r+1)}) \leftarrow \mathbf{Gen}^{\mathbf{Sig}}$ and pseudonym $p_{(r,v_r+1)} \leftarrow \mathbf{Gen}^{\mathbf{Pse}}$
- Replace $(r, -, -, -, v_r)$ with $(r, p_{(r,v_r+1)}, \mathbf{k}^{\mathbf{enc}}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{ver}}_{(r,v_r+1)}, v_r + 1,)$ in $\mathbf{R_c}$
- Find $(A, r, c, v_{(r,v_r)}) \in \mathbf{UR_c}$
- Decrypt $\left(-, \mathbf{k}^{\mathbf{dec}}_{(r,v_r)}, -, -\right) = \mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_A}(c)$
- 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 $\mathbf{UR_t}$
- $\forall (r, f, -, -, -, v_f, -, op) \in \mathbf{PA_c}$:
  * Find $(f, -, v_f, Enf) \in \mathbf{F_c}$
  * If $\mathsf{Enf_c} \in Enf$:
    · Generate new symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f+1)} \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$
    · Set $c_r = \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r,v_r+1)}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,v_f+1)}\right)$
  * Else:
    · Set $\mathbf{k}^{\mathbf{sym}}_{(f,v_f+1)} = \mathbf{N}$
    · Set $c_r = \mathbf{N}$
  * Generate new pseudonym $p_{(f,v_f+1)} \leftarrow \mathbf{Gen}^{\mathbf{Pse}}$ for f
  * Replace $(f, -, v_f, -)$ with $(f, p_{(f,v_f+1)}, v_f + 1, -)$ in $\mathbf{F_c}$
  * Add $(r, f, p_{(r,v_r+1)}, p_{(f,v_f+1)}, c_r, v_f + 1, v_r + 1, op)$ to $\mathbf{PA_c}$
  * For all $(r', f, p_{(r',v_{r'})}, -, -, -, v_{r'}, op') \in \mathbf{PA_c}$ *s.t.* $r' \neq r$:
    · Find $(r', -, \mathbf{k}^{\mathbf{enc}}_{(r',v_{r'})}, -, -) \in \mathbf{R_c}$
    · If $\mathsf{Enf_c} \in Enf$:
      ◦ Set $c_{r'} = \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r',v_{r'})}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,vf+1)}\right)$
    · Else:
      ◦ Set $c_{r'} = \mathbf{N}$
    · Add $(r', f, p_{(r',v_{r'})}, p_{(f,v_f+1)}, c_{r'}, v_f + 1, v_{r'}, op')$ to $\mathbf{PA_c}$
  * Delete all $(-, f, -, -, -, v_f, -, -)$ to $\mathbf{PA_c}$
  * Publish $(f, p_{(f,v_f+1)}, v_f + 1, Enf)$ as a retained message to f

$\underline{revokePermissionFromRole_A(r, \langle f, op \rangle)}$

- Find $(r, f, -, -, -, -, -, op'') \in \mathbf{PA_c}$
- If $op'' \subseteq op$:
  * Find $(f, -, v_f, Enf) \in \mathbf{F_c}$
  * If $\mathsf{Enf_c} \in Enf$:
    · Generate new symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f+1)} \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$
  * Else:
    · Set $\mathbf{k}^{\mathbf{sym}}_{(f,v_f+1)} = \mathbf{N}$
  * Generate new pseudonym $p_{(f,v_f+1)} \leftarrow \mathbf{Gen}^{\mathbf{Pse}}$ for f
  * For all $(r', f, -, -, -, v_f, -, op') \in \mathbf{PA_c}$ *s.t.* $r' \neq r$:
    · Find $(r', -, \mathbf{k}^{\mathbf{enc}}_{(r',v_{r'})}, -, -) \in \mathbf{R_c}$
    · If $\mathsf{Enf_c} \in Enf$:
      ◦ Set $c_{r'} = \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{enc}}_{(r',v_{r'})}}\left(\mathbf{k}^{\mathbf{sym}}_{(f,v_f+1)}\right)$
    · Else:
      ◦ Set $c_{r'} = \mathbf{N}$
    · Add $(r', f, -, p_{(f,v_f+1)}, c_{r'}, v_f + 1, -, op')$ to $\mathbf{PA_c}$
  * Delete all $(-, f, -, -, -, v_f, -, -)$ to $\mathbf{PA_c}$
  * Delete $(r, \langle f, op' \rangle)$ from $\mathbf{PA_t}$
  * Replace $(f, -, v_f, -)$ with $(f, p_{(f,v_f+1)}, v_f + 1, -)$ in $\mathbf{F_c}$
  * Publish $(f, p_{(f,v_f+1)}, v_f + 1, Enf)$ as a retained message to f
- 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 3: Role-based Cryptographic Access Control for Data in Transit - Administrative Actions

$\underline{init_u()}$

- Generate encryption key pair $(\mathbf{k}^{\mathbf{enc}}_u, \mathbf{k}^{\mathbf{dec}}_u) \leftarrow \mathbf{Gen}^{\mathbf{Pub}}$, signature key pair $(\mathbf{k}^{\mathbf{ver}}_u, \mathbf{k}^{\mathbf{sig}}_u) \leftarrow \mathbf{Gen}^{\mathbf{Sig}}$ and pseudonym $p_u \leftarrow \mathbf{Gen}^{\mathbf{Pse}}$
- Replace $(u, \mathbf{N}, \mathbf{N}, \mathbf{N})$ with $(u, p_u, \mathbf{k}^{\mathbf{enc}}_u, \mathbf{k}^{\mathbf{ver}}_u)$ in $\mathbf{U_c}$

$\underline{readResource_u(f)}$

- If $\exists(r, -, -, -, v_r) \in \mathbf{R_c}\, s.t. (u, r, c, v_r) \in \mathbf{UR_c} \wedge (r, f, -, -, c_r, v_f, -, op) \in \mathbf{PA_c} \wedge \mathsf{Sub} \in op$:
  - ∗ Subscribe to f
  - ∗ Upon receiving a message fc_{enc} from f, considering the latest retained message $(f, -, v_f, Enf)$ in f:
    - · If $\mathsf{Enf_c} \in Enf$:
      - ∘ Decrypt $\left(-, \mathbf{k}^{\mathbf{dec}}_{(r,v_r)}, -, -\right) = \mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_u}(c)$
      - ∘ Decrypt $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)} = \mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_{(r,v_r)}}(c_r)$
      - ∘ Decrypt $fc = \mathbf{Dec}^{\mathbf{S}}_{\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}}(fc_{enc})$
    - · Else:
      - ∘ Set $fc = fc_{enc}$
    - · Return fc
- Else:
  - ∗ Return $\perp$

$\underline{writeResource_u(f, fc)}$

- If $\exists(r, -, -, -, v_r) \in \mathbf{R_c}\ s.t.\ (u, r, c, v_r) \in \mathbf{UR_c} \wedge (r, f, -, -, c_r, -, -, op) \wedge \mathsf{Pub} \in op$:
  - ∗ Find $(f, -, v_f, Enf) \in \mathbf{F_c}$
  - ∗ If $\mathsf{Enf_c} \in Enf$:
    - · Decrypt $\left(-, \mathbf{k}^{\mathbf{dec}}_{(r,v_r)}, -, -\right) = \mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_u}(c)$
    - · Decrypt $\mathbf{k}^{\mathbf{sym}}_{(f,v_f)} = \mathbf{Dec}^{\mathbf{P}}_{\mathbf{k}^{\mathbf{dec}}_{(r,v_r)}}(c_r)$
    - · Encrypt $fc_{enc} = \mathbf{Enc}^{\mathbf{S}}_{\mathbf{k}^{\mathbf{sym}}_{(f,v_f)}}(fc)$
  - ∗ Else:
    - · Set $fc_{enc} = fc$
  - ∗ Publish fc_{enc} as a message to f
- Else:
  - ∗ Return $\perp$

Figure 4: Role-based Cryptographic Access Control for Data in Transit - User Actions

which are known by all authorized users, as presented in Section 6.2. However, since users authenticates toward the broker, the broker itself can provide accountability. If necessary, users can also be required to digitally sign published messages using their private signing keys, at the cost of increasing computational overhead. Modifying the CAC scheme to accommodate this behavior would be straightforward, and can be seen as an instance of the AC model introduced in [7, 8] that considers the features of the client used by a user to access a certain resource; despite a user (e.g., a general practitioner) can be entitled to access a resource containing sensitive data (e.g., the healthcare information of a patient), it can be denied such a permission because of the low level of protection offered by the client (e.g., personal smartphone) or it can be granted when an adequate client is used (e.g., desktop operated by the hospital).

On Cryptography. At a high level, we assume cryptographic primitives to be perfect. Consequently, the confidentiality and the integrity of encrypted data cannot be violated except by (computationally infeasible) brute force attacks. Moreover, the use of AEAD — which is a more robust and secure variant of authenticated encryption — allows binding a ciphertext to the context where it is supposed to be used, thus avoiding replay attacks at the CAC scheme level. On the other hand, protection against replay attacks at the application level (e.g., those in Sections 4.1 to 4.3) must be provided by, e.g., dedicated challenge-response protocols. In other words, our CAC scheme guarantees confidentiality and integrity of sensitive data while enforcing AC policies ; however, how the data are then used by the application is out of the scope of this work. For a discussion on the security of the chosen cryptographic algorithms, please refer to Section 6.4.

On Domains and Secure Key Distribution. The presence of the four security domains — and the different placement of entities into domains (i.e., architectures) — does not affect the inner functioning of our CAC scheme. In other words, key distribution is carried out as described in Figures 3 and 4 regardless of the chosen architecture. Also, as shown in Table 4, we note that $\mathbf{P_c}$ contains public information (e.g., public keys) or encrypted information (e.g., encrypted private and symmetric keys) only. By construction, even though $\mathbf{P_c}$ is public, only authorized users can decrypt roles' private keys and, consequently, access resources' symmetric keys. Indeed, the secure key distribution of resources' symmetric keys and roles' private keys is guaranteed by using key transport as defined by the NIST [9]: one party (in our case, the administrator) selects and encrypts keys to then distribute them to other parties (in our case, the users). More precisely, resources' symmetric keys are encrypted with the public keys of authorized roles, while the roles' private keys are encrypted with the public keys of authorized users.

On Revocations. Adding permissions to $\mathbf{P_c}$ is straightforward and consists in encrypting private information (i.e., secret or private keys) with the public key of the newly authorized roles or users. On the other hand, revoking permissions from $\mathbf{P_c}$ requires careful management of cryptographic material. To be conservative, we consider the worst-case scenario in which a user u previously cached all secret keys — both of roles and resources — the user could access to. Hence, when revoking permissions from a user u (or from one of the roles that u is assigned to), the administrator needs to renew all the involved secret keys. More precisely, when revoking a permission $\langle f, op \rangle$ — where f has version number v_f — from a role r (i.e., when invoking the action $\underline{revokePermissionFromRole_A(r, \langle f, op \rangle)}$) in our CAC scheme, we distinguish two cases:

- if r already had a permission $\langle f, op' \rangle$ so that $op' \subseteq op$, the administrator generates a new symmetric key $\mathbf{k}^{\mathbf{sym}}_{(f,v_f+1)} \leftarrow \mathbf{Gen}^{\mathbf{Sym}}$ for f, which the administrator distributes to all *other* authorized roles. In this way, users belonging to r do not have access to the new symmetric key;
- if r already had permission op' so that $op' \cap op \neq \emptyset$, the administrator simply updates $\mathbf{PA_t}$ and $\mathbf{PA_c}$ by removing the actions in op from op'.

Similarly, when revoking a user u from a role r with version number v_r (i.e., when invoking the action $\underline{revokeUserFromRole_A}(u, r)$), the administrator generates new keys $(\mathbf{k}^{\mathbf{enc}}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{dec}}_{(r,v_r+1)}) \leftarrow \mathbf{Gen}^{\mathbf{Pub}}$ and $(\mathbf{k}^{\mathbf{ver}}_{(r,v_r+1)}, \mathbf{k}^{\mathbf{sig}}_{(r,v_r+1)}) \leftarrow \mathbf{Gen}^{\mathbf{Sig}}$ for r, and distributes them to all *other* authorized users. In this way, u does not have access to r's new private keys. Then, the administrator also renews the symmetric keys of all resources r had access to with the $\underline{revokePermissionFromRole_A(r, \langle f, op\rangle)}$ action.

6.4. Implementation

We implement the pseudocode presented in Figures 3 and 4 in a tool named CryptoAC.[21] We choose the Kotlin multiplatform[22] programming language for its intrinsic portability and the possibility of a native deployment in IoT devices — avoiding the computational overhead of a Java virtual machine (JVM). This is especially relevant since Kotlin mainly targets Linux-based environments,[23] which are the most deployed in IoT devices.[24] CryptoAC can act both as an MQTT client — by using the Eclipse Paho library[25] — and also as an administrative tool. In other words, CryptoAC can be used by both IoT clients and the administrator for managing $\mathbf{P_c}$ and $\mathbf{P_t}$ through the set of actions in Table 3. CryptoAC uses the Ktor library[26] to expose RESTful APIs and a web interface developed with Kotlin/JS and the React library.[27] All APIs' inputs are validated with OWASP-approved regular expressions[28] to avoid web-based attacks (e.g., injection, Cross-Site Scripting). We choose the Mosquitto MQTT broker[29] as DM, enabling the DYNSEC plugin (see Section 3.2) allowing for traditional enforcement of RBAC policies — depending on the enforcement type(s) *Enf* chosen for each topic. Then, as in [24], we choose Redis[30] to store metadata, that is, the two policy states $\mathbf{P_t} = \langle \mathbf{U_t}, \mathbf{R_t}, \mathbf{F_t}, \mathbf{UR_t}, \mathbf{PA_t}\rangle$ and $\mathbf{P_c} = \langle \mathbf{U_c}, \mathbf{R_c}, \mathbf{F_c}, \mathbf{UR_c}, \mathbf{PA_c}\rangle$. Redis is primarily an in-memory storage, a characteristic that allows for low response time to queries. Moreover, the (dedicated) protocol used by Redis (i.e., RESP, REdis Serialization-Protocol) is geared toward low latency. The metadata of each user are stored under a unique Redis key, while a list collects all users' Redis keys. We follow the same approach for the metadata of roles, topics, user-role, and role-permission pairs. Access to the Redis datastore is protected by individual passwords. Finally, the upper bound in size for the identifiers and the pseudonyms of users, roles, and resources is 50 bytes, for version numbers are at most 8 bytes (i.e., 2^{64} values) and for the flags *Enf* and *op* are 1 byte. More details on the configuration of Redis can be found in the dedicated documentation.[31]

Cryptographic Algorithms. As cryptographic provider, we choose Sodium,[32] a modern and portable cryptographic library whose implementation was thoroughly audited and no flaws or vulnerabilities were found.[33] Sodium supports AEAD, whose usage is in line with the requirements contained in the call for Lightweight Cryptography to protect small electronics — thus including IoT devices — issued by the NIST.[34] In particular, Sodium supports two popular cryptographic algorithms for AEAD: 256-bit AES-GCM with, e.g., the SHA-384 hash function (i.e., AES-256-GCM) and the XChaCha20 symmetric stream cipher with 192-bit nonce extension and the Poly1305 universal hash function (i.e., XChaCha20-Poly1305) — these are the same cryptographic algorithms suggested in TLS v1.3 [56]. Between the two, we choose the latter because, although hardware acceleration for AES is often available in modern processors, its performance on platforms that lack such hardware is considerably lower (e.g., not all IoT devices are equipped with hardware acceleration for AES). More importantly, another issue is that many software-only AES implementations are vulnerable to cache-collision timing attacks [17]. Instead, XChaCha20-Poly1305 is faster than (non-accelerated) AES and provides homogeneous performance across similar hardware, enhancing portability. Finally, we note that XChaCha20-Poly1305 expects keys of fixed size, i.e., 256 bits.

Concerning the cryptographic algorithm for en/decryption of keys, Sodium supports only ECDH-X25519 — that is, Elliptic-Curve Diffie-Hellman key exchange using the Curve25519 elliptic curve. The Curve25519 curve has high performance and strong security properties, is free from known patents, and is in fact recommended by both NIST [23] and RFC 7748 [46]. Finally, we note that ECDH-X25519 expects keys of fixed size, i.e., 256 bits (offering 128 bits of security). Alternatives to Curve25519 exist and, the higher the key size, the more secure and less performance they provide (e.g., Curve448 expects keys of fixed size 448 offering 224 bits of security).

Finally, concerning the cryptographic algorithm for digital signatures, Sodium supports only EdDSA-Ed25519 — that is, for Edwards-curve Digital Signature Algorithm using the Ed25519 elliptic curve. Note that Ed25519 actually uses the bi-rational equivalent twisted Edwards form of Curve25519 ($y^2 = x^3 + 486662x^2 + x$ over $\mathbb{F}_{2^{255}-19}$) but use different coordinate representations: Curve25519 in Montgomery form (optimized for key exchange — X25519) and Ed25519 in twisted Edwards form (optimized for signatures — EdDSA-Ed25519). Hence the same considerations done for ECDH-X25519 (e.g., recommendations, key sizes, trade-off between security and performance) apply also to EdDSA-Ed25519.

[21] https://github.com/stfbk/CryptoAC

[22] https://kotlinlang.org/docs/multiplatform.html

[23] https://www.jetbrains.com/lp/devecosystem-2019/

[24] https://outreach.eclipse.foundation/iot-edge-developer-2021

[25] https://www.eclipse.org/paho/

[26] https://ktor.io/

[27] https://it.reactjs.org/

[28] https://owasp.org/www-community/OWASP_Validation_Regex_Repository

[29] https://hub.docker.com/_/eclipse-mosquitto

[30] https://redis.io/

[31] https://cryptoac.readthedocs.io/

[32] https://libsodium.gitbook.io/doc/

[33] https://www.privateinternetaccess.com/blog/libsodium-audit-results/

[34] https://www.nist.gov/news-events/news/2018/04/nist-issues-first-call-lightweight-cryptography-protect-small-electronics.

Toward a Production-Ready Deployment. While carefully implemented, CryptoAC is obviously still a prototype and not ready for production. The main areas that need further development and implementation effort are the following. First, a robust key management system should be implemented to address the whole lifecycle of cryptographic keys (e.g., key generation, registration, storage, and archival), possibly adopting the guidelines of the NIST in [9]. Moreover, comprehensive testing, penetration testing, and management of the software bill of materials should be conducted to ensure reliability and continuous vulnerability monitoring. Similarly, compliance and auditing should be assessed to guarantee adherence to security standards (e.g., NIST FIPS[35]) and regulations (e.g., GDPR[36]). Finally, comprehensive documentation and compatibility with different platforms should be considered to provide seamless integration with third-party systems.

7. Optimization Problem Extension

As introduced in Section 5.2, we now present the optimization problem for identifying CAC scheme architectures that simultaneously minimize risk levels — derived from the trust assumptions — and maximize the quality of service — based on the performance goals — of a given scenario. Below, we first enumerate all possible architectures for CAC schemes deployed in Cloud-Edge-IoT applications (Section 7.1); this amounts at defining the search space of the optimization problem. Then, we explain how to determine goals scores (i.e., integer numbers) for each architecture (Section 7.2), and how to calculate risk levels depending on the trust assumptions of the considered scenario (Section 7.3). Finally, we show how to combine goal scores and risk levels into an optimization problem (Section 7.4) for which we present a proof-of-concept application in Section 8.

As explained at the end of Section 1, we build our contributions on top of an already existing work considering CAC schemes in Cloud applications (see [12]). In detail, we extend it to applications involving also the Edge and the IoT. In each section below, we first briefly report the contributions of the original work in [12], and then — in a dedicated paragraph — our novel contributions; this allows for more clearly distinguish between existing work and novel contributions. For more details on the original work, we suggest the interested reader to refer to [12].

7.1. Architectural Model

The authors in [12] present an architectural model identifying 81 architectures for CAC schemes in Cloud applications by considering possible assignments between the set of entities E = {proxy, RM, MM, DM} and the set of domains D = {client, on-premise, Cloud} (see Section 5.1). *Possible assignments* means that certain entity-domain assignments are automatically excluded a-priori. For instance, assigning the proxy entity — which performs all cryptographic computations — to the Cloud domain would ideally allow the Cloud provider to access the secret cryptographic keys contained and used within the proxy. With these keys, the Cloud provider could decrypt stored data by itself, thus breaching the confidentiality of the data.

Architectural Model Extension. We extend the aforementioned architectural model in two directions: adding the Edge to the available domains and removing limitations on possible assignments.

First, we extend D by adding the Edge domain, i.e., $D_{ext} = D \cup \{\text{Edge}\}$. We define the Edge as a domain offering computing, network, and storage services geographically distributed at the outskirts of the network. Differently from the Cloud, the Edge has lower computational capabilities but is closer to services and users, yielding lower latency. The Edge can be operated by organizations through third-party software (e.g., AWS IoT Greengrass[37]) or by partially trusted providers (e.g., Google Distributed Cloud[38]).

Second, we argue that the limitations on possible entity-domain assignments may be bypassed by using technologies such as TEEs. In brief, a TEE (e.g., ARM TrustZone[39]) is a secure area of the processor that guarantees confidentiality and integrity of data and instructions through isolated execution. Noticeably, TEEs are becoming widely available in major Cloud and Edge providers (e.g., Google[40], Azure[41]). In CAC, TEEs allow for using cryptographic keys securely, thus permitting to assign the proxy to the Cloud domain [27]. Generalizing, we do not preclude any entity-domain assignment a-priori, obtaining as a result a simpler and more comprehensive architectural model. In detail, we consider an extended set of architectures $\mathcal{ARC}_{ext} = \mathcal{P}(D_{ext} \times E)$ — where "$\mathcal{P}$" is the power set operator — which contains 65,536 (i.e., $2^{(4 \cdot 4)}$) different architectures. Note that architectures may assign the same entity to multiple domains at the same time; we call these *hybrid architectures*.

Intuitively, administrators can still easily filter out entity-domain assignments not suitable to their scenario by specifying a set of prefilters $PRE \subseteq (D_{ext} \times E)$, as mentioned in Section 5.2; in other words, prefilters concur in defining the search space of the optimization problem. Consequently, the subset of architectures considered in the optimization problem is $\mathcal{ARC}_{sub} = \{arc \in \mathcal{ARC}_{ext} \ : \ (arc \cap PRE = \emptyset)\}$.

7.2. Goals Evaluation

The authors in [12] identify 8 goals relevant to the architecture of Cloud-based applications: redundancy, scalability, reliability, maintenance, DoS resilience, minimization of Cloud vendor Lock-in, on-premise monetary savings and Cloud monetary savings. As also mentioned in Section 5.2, the authors

[35] https://csrc.nist.gov/publications/fips
[36] https://gdpr.eu/
[37] https://aws.amazon.com/greengrass/
[38] https://cloud.google.com/distributed-cloud
[39] https://developer.arm.com/ip-products/security-ip/trustzone
[40] https://cloud.google.com/confidential-computing
[41] https://azure.microsoft.com/en-us/solutions/confidential-compute/

then measure how a CAC architecture $arc \in \mathcal{ARC}_{sub}$ attains a certain goal by considering how each entity-domain assignment of *arc* affects that goal in isolation. In general, each entity-domain assignment has either a positive (+), negligible or minimal (=), or negative (-) influence on each goal. For instance, deploying the proxy entity in the $client_u$ domain (i.e., in the devices of the user u) has a positive influence (i.e., +) on the scalability of the architecture — as cryptographic computations are distributed among client devices. Conversely, deploying (a single instance of) the proxy in the on-premise domain has a negative influence (i.e., -) on scalability — as the proxy would easily become a bottleneck. The authors in [12] define the influence that each possible assignment has on goals by analyzing relevant academic literature and industrial technical reports. To then determine how *arc* attains a given goal, the authors combine the influences of all entity-domain assignments of *arc* by adding together the +, - and = symbols as adding the +1, -1 and 0 numbers, respectively. Formally, this is equivalent to considering an objective function $g : \mathcal{ARC}_{sub} \mapsto \mathbb{Z}$ associated with each goal; having as input an architecture (i.e., a set of entity-domain assignments), an objective function for a goal returns the sum of the influences of all entity-domain assignments of the architecture on that goal. By repeating the same process for all objective functions, it is possible to obtain an array of goal scores for each architecture.

Goals Evaluation Extension. We extend the aforementioned goals evaluation by considering how the assignment of entities to the Edge domain affects the goals presented in [12], and report the influences in Table 5:

- *Redundancy*: Cloud computing can rely on dedicated servers for guaranteeing redundancy of services and resources. Conversely, the Edge can rely on other Edge nodes only — possibly already in use by other tenants — to act as redundant servers, resulting in a limited redundancy capacity [67];
- *Scalability*: the ability of the Edge to scale up and down to accommodate dynamic workloads is determined by the amount of (e.g., computational and memory) resources available which — according to [52, 42, 67] — is comparable to the Cloud;
- *Reliability*: compared to Cloud computing, the Edge has a higher chance of failure of devices and nodes [52]. Moreover, it is generally more challenging to provide fault tolerance for Edge services [42];
- *Maintenance*: the Edge is often operated by a third party, hence deployment (e.g., setup and configuration of the infrastructure) and maintenance (e.g., operative systems and runtime environments updates) are usually not a concern;
- *DoS Resilience*: the Cloud is generally more resistant to DoS attacks than the Edge. In turn, the Edge is generally more resistant to DoS attacks than on-premise data centers [21, 52];
- *Minimization of Cloud Vendor Lock-in*: assigning entities to the Edge domain does not affect the Cloud vendor lock-in goal;
- *On-premise Monetary Savings*: assigning entities to the Edge domain does not affect the on-premise monetary savings goal;
- *Cloud Monetary Savings*: assigning entities to the Edge domain does not affect the Cloud monetary savings goal.

Table 5: Entity-Domain Influence on Goals

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	+	-	=	+	+	-	=	+	+	-	=	+	+	-	=	+
Min. Cloud Vendor Lock-in	+	+	+	-	+	+	+	-	+	+	+	-	+	+	+	-
On-premise Monetary Savings	+	-	+	+	+	-	+	+	+	-	+	+	+	-	+	+
Cloud Monetary Savings	+	+	+	-	+	+	+	-	+	+	+	-	+	+	+	-
Bandwidth	-	-	=	+	-	-	=	+	-	-	=	+	-	-	=	+
Latency	=	=	+	-	=	=	+	-	=	=	+	-	=	=	+	-
Computational Power	-	-	=	+	-	-	=	+	-	-	=	+	-	-	=	+
Memory	-	-	-	+	-	-	-	+	-	-	-	+	-	-	-	+
Min. Edge Vendor Lock-in	+	+	-	+	+	+	-	+	+	+	-	+	+	+	-	+
Edge Monetary Savings	+	+	-	+	+	+	-	+	+	+	-	+	+	+	-	+

Besides the 8 goals identified in [12] (relevant to Cloud-based applications), we identify 6 new goals relevant to Edge-IoT applications (see Table 5):

- *Bandwidth* [52, 18, 42, 67]: that is, the maximum rate of data transfer over (a given path of) a network. High bandwidth is a peculiarity of Cloud computing [26] whereas, being at the outskirts of the network, the Edge usually has lower — but still considerable, at least with respect to on-premise infrastructures — bandwidth [18, 52];
- *Latency* [63, 18, 52, 67, 42]: that is, the delay in communicating a bit of data across a network. Edge computing is renowned for its lower latency with respect to the Cloud [67, 52], which is generally even less accessible than on-premise data centers [63];
- *Computational Power* [67, 42, 52]: that is, the ability to process incoming data rapidly. Both the Edge and the Cloud have large computational power, although some Edge nodes — similar to on-premise data centers — may be more resource-constrained [52, 42];
- *Memory* [67, 42, 52]: that is, the amount of primary memory (i.e., RAM). Similarly to the computational power goal, the Cloud is more resourceful than Edge and on-premise data centers [42, 67];

- *Minimization of Edge Vendor Lock-in* [43, 64]: that is, the easiness in switching Edge provider (e.g., from Google Distributed Cloud to AWS Greengrass). Intuitively, each entity in the Edge domain stresses the vendor lock-in effect;
- *Edge Monetary Savings* [45, 52, 64, 42]: that is, the monetary savings due to *not* adding entities to the Edge domain. Intuitively, the fewer entities run in the Edge, the more Edge-related costs are reduced.

As a final remark, we note that organizations can easily fine-tune the influence of entity-domain assignments on goals based on their specific context. In other words, being extracted from the literature, the influences shown in Table 5 cover a wide variety — but not all possible — contexts. Moreover, as said in [12], the extended set of goals is not meant to be exhaustive or representative of all scenarios and applications; further goals can be easily added.

7.3. Risk Assessment Methodology

As discussed in Section 5.2, sensitive data in Cloud-Edge-IoT applications are subject to a heterogeneous set of attackers (i.e., external attackers, malicious insiders, partially trusted providers). In the context of CAC, these attackers may compromise the confidentiality, integrity, and availability of the sensitive data by targeting (the secret keys, metadata, and encrypted data of) CAC schemes. For this reason, the authors in [12] propose a risk assessment methodology to evaluate the risk exposure of CAC architectures to these attackers where — as typically done [36] — the risk is decomposed into two dimensions, i.e., likelihood and impact, each of which can be either negligible, low, medium, or high. To preserve the generality of the approach, the authors group all possible low-level threats (e.g., spoofing, eavesdropping, session hijacking, spear phishing, side-channel attacks, device cloning) under a single attacker for each domain and communication channel between pairs of domains. The likelihood of each attacker is derived from the trust assumptions of the underlying scenario (see Sections 4.1 to 4.3); if unspecified, the likelihood of an attacker is set to a default value (e.g., medium). Then, the authors precompute the impact of these attackers based on what information they can (potentially) compromise. For instance, an attacker compromising the proxy entity would have access to the secret keys used for en/decryption in CAC (violating the confidentiality and the integrity of sensitive data), while an attacker compromising the DM entity could tamper with encrypted data or perform a DoS attack (i.e., violating the integrity and the availability of sensitive data). Finally, the impact and the likelihood are combined using a risk table (i.e., see [36]) to determine the overall risk levels — either negligible, low, medium, or high — on the confidentiality, integrity, and availability of sensitive data. Similarly to Section 7.2, this is equivalent to considering three objective functions g_C, g_I, g_A. As the final purpose is to maximize the values of objective functions (see Section 7.4), the authors in [12] define g_C, g_I and g_A as to measure the complement of the risk levels, i.e., the protection levels: the authors simply invert the scale of risk levels and associate a number from 0 (negligible protection, high-risk level) to 3 (high protection, negligible risk level).

Risk Assessment Methodology Extension. We extend the aforementioned risk assessment methodology by adding the Edge domain — and the corresponding attacker — to the risk assessment methodology. Moreover, we also add one communication channel between the Edge and each other domain — and the corresponding attackers — to the risk assessment methodology.

7.4. Multi-Objective Combinatorial Optimization Problem

Once defined 14 objective functions for goals — 8 from [12] and 6 proposed in Section 7.2 — and 3 objective functions for protection levels (see Section 7.3), we can define the optimization problem: given the set of architectures $\mathcal{ARC}_{sub}$, we formalize as a Multi-Objective Combinatorial Optimization Problem (MOCOP) [44] the problem of finding the architecture for which the tuple $(g_{\text{latency}}, \ldots, g_C, g_I, g_A)$ of objective functions is maximum (that is, the architecture that simultaneously maximizes all objective functions):

$$\max_{arc \in \mathcal{ARC}_{sub}} (g_{\text{latency}}(arc), \ldots, g_C(arc), g_I(arc), g_A(arc)); \tag{1}$$

We note that there may be multiple equally good solutions (i.e., CAC architectures, in our case) that simultaneously maximize all objective functions in Equation (1); these solutions are called *Pareto Optimal* [44]. Formally, an architecture $arc^* \in \mathcal{ARC}_{sub}$ is Pareto Optimal with respect to the MOCOP in Equation (1) if and only if there is no other architecture $arc \in \mathcal{ARC}_{sub}$ such that $g_i(arc) \geq g_i(arc^*)$ for each objective function g_i and $g_j(arc) > g_j(arc^*)$ for at least one objective function g_j. In other words, an architecture is Pareto Optimal if there does not exist another architecture that improves one objective function without detriment to another; as in [35], we write $arc^* \succ arc$ to denote that arc^* is Pareto Optimal with respect to arc (see Table 6 for some examples of Pareto Optimal architectures). The set of Pareto Optimal architectures can be identified by solving the problem in Equation (1) reusing off-the-shelf approaches available in the literature. For instance, a straightforward approach is to treat the MOCOP as a multi-dimensional maximum vector problem and solve it using the `Best` algorithm described in [35] (and shown in Algorithm 1).

Reduction to Constrained Single-Objective Optimization. The final choice of which architecture to select among those Pareto Optimal is ultimately left to the organization; this may require some ingenuity by knowledgeable administrators, that should meet to reach a consensus on which architecture to choose based on criteria that are difficult to formalize as objective functions to be included in the MOCOP. A possible alternative — that we describe below — is to reduce the MOCOP to a Constrained Single-Objective Optimization Problem (CSOOP): to do so, administrators assign (*i*) a weight w representing a relative importance or priority to (the performance goal or security property of) each objective function and (*ii*) a constraint — consisting of a threshold t and a penalty p — to the value

Table 6: Examples of Pareto Optimal Architectures with Respect to Equation (1) (scores go from -4 to +4)

Domain: Client	Domain: On-premise	Domain: Edge	Domain: Cloud	Redundancy	Scalability	Reliability	Maintenance	DoS Resilience	Min. Cloud Vendor Lock-in	On-premise Monetary Savings	Cloud Monetary Savings	Bandwidth	Latency	Computational Power	Memory	Min. Edge Vendor Lock-in	Edge Monetary Savings -	Protection Level C	Protection Level I	Protection Level A
				-1	+2	-1	+4	+2	+2	+4	+2	0	+1	0	-2	0	0	0	+3	0
				+2	+2	+2	+2	+2	-2	+2	-2	+2	-3	+2	+2	+4	+4	0	+3	0
				-3	0	-3	+3	0	+3	+3	+3	0	+2	0	-3	-1	-1	0	+3	0
				0	+3	0	+4	+3	+1	+4	+1	+1	0	+1	-1	+1	+1	0	+3	0

Algorithm 1: Best algorithm from [35]

Input: $\mathcal{ARC}_{sub}$
Output: $\mathcal{ARC}_{opt}$ `// Set of Pareto Optimal Solutions`

$\mathcal{ARC}_{opt} = \emptyset$;
while $\mathcal{ARC}_{sub} \neq \emptyset$ **do**
  $arc^* = \text{pop}(\mathcal{ARC}_{sub})$;
  `// Find a Pareto Optimal solution`
  foreach $arc \in \mathcal{ARC}_{sub}$ **do**
    if $(arc > arc^*)$ **then**
      $arc^* = arc$;
  end
  $\mathcal{ARC}_{opt} = \mathcal{ARC}_{opt} \cup \{arc^*\}$;
  `// Remove architectures not Pareto Optimal w.r.t` arc^*
  foreach arc *in* $\mathcal{ARC}_{sub}$ **do**
    if $(arc^* > arc)$ **then**
      $\mathcal{ARC}_{sub} = \mathcal{ARC}_{sub} \setminus \{arc\}$;
  end
end
return $\mathcal{ARC}_{opt}$;

Algorithm 2: Algorithm for Solving the CSOOP

Input: $\mathcal{ARC}_{sub}, w_{\text{latency}}, t_{\text{latency}}, p_{\text{latency}}, \ldots, w_{\text{A}}, t_{\text{A}}, p_{\text{A}}$
Output: $\mathcal{ARC}_{opt}$ `// Set of Pareto Optimal Solutions`

$\mathcal{ARC}_{opt} = \emptyset$;
$s_{max} = -\infty$;
foreach arc *in* $\mathcal{ARC}_{sub}$ **do**
  $s_{arc} = (\, w_{\text{latency}} \cdot [g_{\text{latency}}]^{p_{\text{latency}}}_{t_{\text{latency}}}(arc) + \ldots + w_{\text{A}} \cdot [g_{\text{A}}]^{p_{\text{A}}}_{t_{\text{A}}}(arc))$;
  if $s_{arc} > s_{max}$ **then**
    $\mathcal{ARC}_{opt} = \{arc\}$;
  else if $s_{arc} = s_{max}$ **then**
    $\mathcal{ARC}_{opt} = \mathcal{ARC}_{opt} \cup \{arc\}$;
end
return $\mathcal{ARC}_{opt}$;

of each objective function. Weights, thresholds, and penalties are positive integers that are determined by the administrators according to the characteristics of the underlying scenario. For instance, if the administrators deem scalability to be twice as important as latency in their scenario, they may set $w_{\text{scalability}} = 2$ and $w_{\text{latency}} = 1$. Similarly, if the administrators want to favor architectures whose scalability's goal score is greater than or equal to a threshold 0, they may set a penalty -5 to every architecture that does not respect the threshold; hence, $t_{\text{scalability}} = 0$ and $p_{\text{scalability}} = -5$. More in general, the semantics is that, given an objective function g, if $g(arc) < t$, then $g(arc)$ is given a penalty value p; formally, we write $[g]^t_p(arc)$ = if $g(arc) < t$ then $g(arc) - p$ else $g(arc)$. The CSOOP then consists in finding the architecture for which the value of the linear combination of all constrained objective functions is maximum:

$$\max_{arc \in \mathcal{ARC}_{sub}} (\, w_{\text{latency}} \cdot [g_{\text{latency}}]^{p_{\text{latency}}}_{t_{\text{latency}}}(arc) + \ldots + w_{\text{A}} \cdot [g_{\text{A}}]^{p_{\text{A}}}_{t_{\text{A}}}(arc)) \quad (2)$$

When solving the CSOOP, administrators can perform a trade-off analysis by fine-tuning the weights and the constraints assigned to each objective function, exploring the space of possible solutions. The final choice of which architecture to select is then trivial — i.e., one among those for which the value computed from Equation (2) is maximum. The corresponding algorithm for solving the CSOOPs is shown in Algorithm 2.

Qualitative vs. Quantitative Objective Functions. The objective functions for goals evaluation (see Section 7.2) and risk assessment (see Section 7.3) provide qualitative scores. Although these qualitative scores are a product of careful analysis and reasoning, some may prefer quantitative scores (i.e., resulting from experimental measurements) for optimization problems. In this regard, we remark that the inner working of objective functions — that is, how objective functions are defined — is independent of the formalization of the optimization problem. In other words, it is possible to redefine the objective functions to make them return quantitative scores. For instance, the authors in [14] already used the optimization problem in [12] to identify the best CAC architecture in an automotive scenario by using quantitative objective functions for measuring latency, and qualitative objective functions for the risk assessment (see [14] for more details). Hence, our optimization problem is flexible enough to accommodate both qualitative and quantitative scores, according to the needs and preferences of organizations.

8. Optimization Problem Application

We now provide a proof-of-concept application of the extended optimization problem described in Section 7 on the three scenarios analyzed in Section 4, i.e., Remote Patient Monitoring, Cooperative Maneuvering, and Smart Lock. The objective of this section is to demonstrate how — starting from a set of different trust assumptions and performance goals — solving the optimization problem allows for identifying each time a different CAC scheme architecture, which is the best according to the considered scenario.

To this end, we implement the Algorithms 1 and 2 — solving the MOCOP in Equation (1) and the CSOOP in Equation (2), respectively — into a dashboard integrated within CryptoAC[42] (see Section 6.4). The dashboard allows administrators to specify prefilters, assign likelihoods to attackers, set constraints, and decide whether to identify optimal architecture(s) by solving the MOCOP or the CSOOP.

8.1. Application to Remote Patient Monitoring

As presented in Section 4.1, we consider the remote monitoring of patients through the use of IoT wearables collecting medical data. We report in Figure 5 a screenshot of the dashboard configured to find the optimal architecture(s) for the Remote Patient Monitoring scenario as explained below.

Concerning the prefilters (see the "Prefilters" card in Figure 5), following *RPM-R2* (use of patients' smartphones as gateways for IoT wearables), we assign the proxy to the client domain. Moreover, *RPM-R5* (hide metadata from partially trusted providers) forces the MM to stay in the on-premise domain. Instead, *RPM-R3* (favor hybrid architectures) makes the DM stay in both the Edge and the Cloud domains. Regarding trust assumptions (see the "Trust Assumptions" card in Figure 5), *RPM-R4* (providers are likely to be honest but curious) sets the likelihood of an attack in the Cloud or the Edge domains to high. Instead, *RPM-R6* (low probability of disgruntled doctors) sets the likelihood of an attacker in the on-premise domain to low. As said in Section 7.3, we set the default likelihood of the other attackers to medium.

As we can see from the "Optimal Architectures" card in Figure 5, solving the MOCOP in Equation (1) for the Remote Patient Monitoring scenario returns 3 Pareto Optimal architectures; the array of numbers in the "Score" column contains the values of the objective functions corresponding to the performance goals in Table 5 — in the same order as in Table 6 — while the last three numbers are for the confidentiality, integrity,

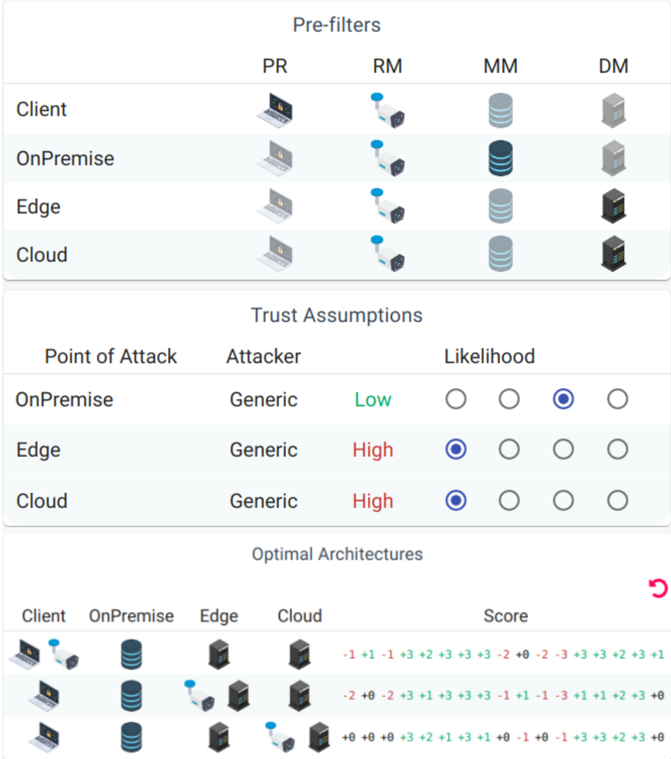

Figure 5: Dashboard for the Remote Patient Monitoring Scenario

and availability objective functions. Once reduced the search space from 65,536 to 3 architectures, administrators can easily choose which to deploy based on other considerations. For instance, they could follow *RPM-R1* (security of data is important) and choose the first architecture since, while still being Pareto Optimal with respect to the other 2 architectures, it has a better score on the availability security property.

8.2. Application to Cooperative Maneuvering

As presented in Section 4.2, we consider CCAM services such as CALM and BSA. We report in Figure 6 a screenshot of the dashboard configured to find the optimal architecture(s) for the Cooperative Maneuvering scenario as explained below.

As prefilters, following *CM-R1* (smart vehicles interact directly with the roadside infrastructure), we set that the proxy stays in the client domain while all other entities can stay in any other domains (i.e., the roadside infrastructure and its backend). Then, *CM-R6* (access to vehicles is protected) lowers the likelihood of an attack in the client domain, while *CM-R4* (public channels) sets the likelihood of communications channels with the client domain to high. Solving the MOCOP with this configuration returns 27 architectures — too many to be compared or analyzed individually. Therefore, we reduce the MOCOP to a CSOOP as described at the end of Section 7.4. For instance, following *CM-R2* (high scalability and ultra-low latency), we could set — in the "Performance Goals and Security Properties" card in Figure 5) — the weights of the scalability and latency goals to 5 (i.e., $w_{\text{scalability}} = w_{\text{latency}} = 5$). Moreover, following *CM-R5* (availability is valuable but not key) and

[42] `https://github.com/stfbk/CryptoAC`

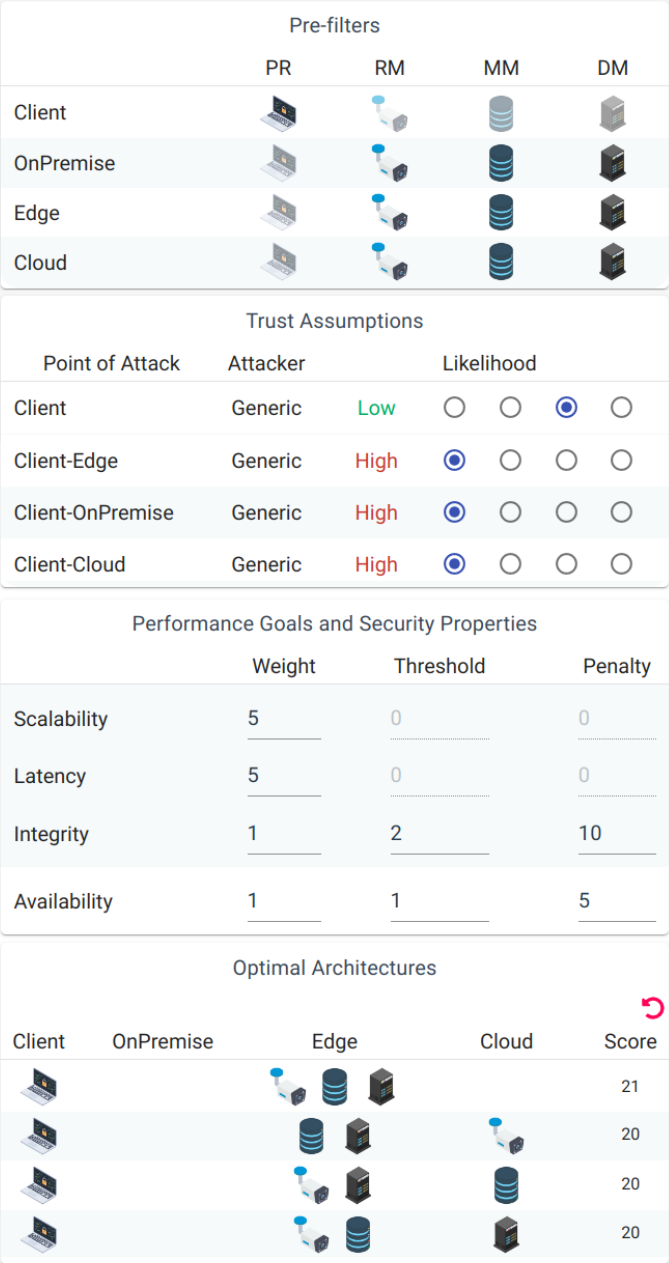

Figure 6: Dashboard for the Cooperative Maneuvering Scenario

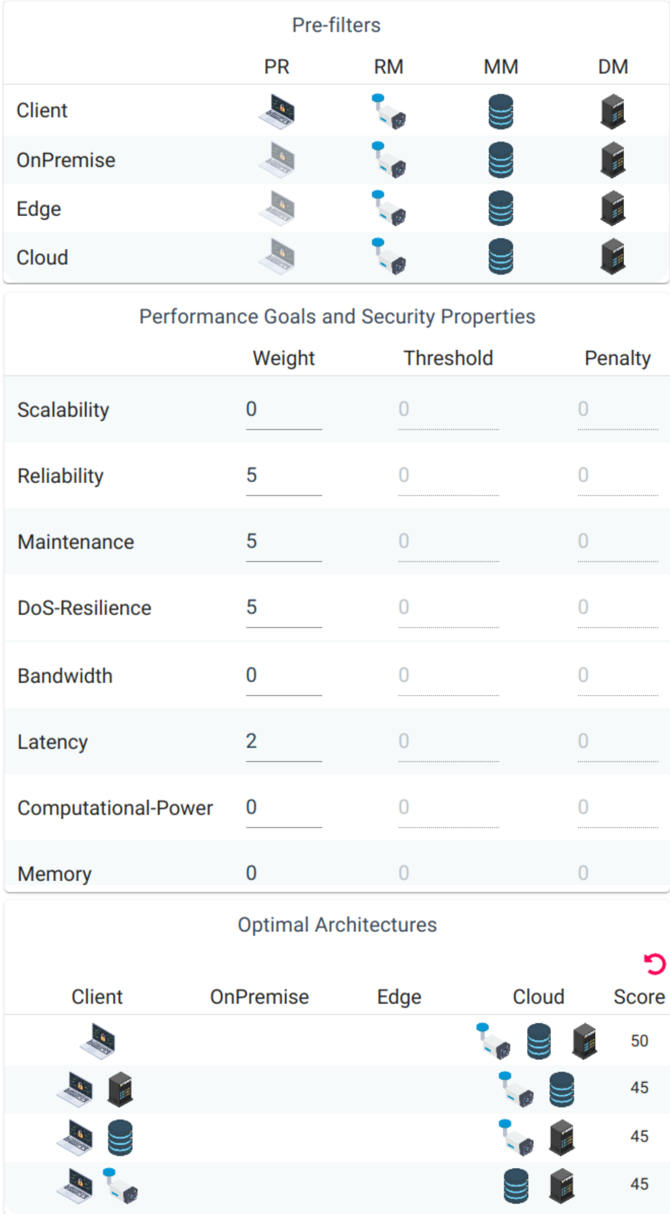

Figure 7: Dashboard for the Smart Lock Scenario

CM-R3 (integrity is fundamental), we could set thresholds and penalties for the availability (e.g., $t_A = 1, p_A = 5$) and the integrity (e.g., $t_I = 2, p_I = 10$) properties. The values of the weights, the thresholds, and the penalties can easily be modified by the administrators to further investigate architectural trade-offs and perform "What-if" analyses on the configuration of the optimization problem.

As we can see from Figure 6, solving the CSOOP in Equation (2) for the Cooperative Maneuvering scenario returns 1 optimal architecture (i.e., the one with the highest score, 21); the number in the "Score" column is the constrained weighted sum of all objective functions. As expected, all entities (except the proxy) are assigned to the Edge domain to reflect the scalability and latency goals. However, the other architectures — which assign one or more entities to the Cloud — perform better on other performance goals and thus achieve a similar score; hence, administrators may still consider them for the deployment of the CAC scheme.

8.3. Application to Smart Lock

As presented in Section 4.3, we consider an organization using smart locks to regulate and restrict access to certain areas (e.g., laboratories, offices) of a building. We report in Figure 7 a screenshot of the dashboard configured to find the optimal architecture(s) for the Smart Lock scenario as explained below.

Without prefilters (except for the proxy in the client domain), the MOCOP returns 185 Pareto Optimal architectures. Therefore, as done in Section 8.2, we reduce the MOCOP to a

CSOOP. First, *SL-R1* gives relevance to the reliability and DoS resilience goals (e.g., we can set $w_{\text{reliability}} = w_{\text{DoS resilience}} = 5$), while *SL-R2* states that low latency is desirable but not crucial (e.g., we can set $w_{\text{latency}} = 2$). Conversely, *SL-R3* and *SL-R4* remark that scalability, memory, bandwidth, and computational power are not relevant goals (e.g., we can set $w_{\text{scalability}} = w_{\text{memory}} = w_{\text{bandwidth}} = w_{\text{computational power}} = 0$). Finally, *SL-R5* requires minimizing the maintenance effort (e.g., we can set $w_{\text{maintenance}} = 5$).

As we can see from Figure 7, solving the CSOOP in Equation (2) for the Smart Lock scenario returns 1 optimal architecture (i.e., the one with the highest score, 50). Intuitively, the choice of deploying many entities in the Cloud reflects the scalability, reliability, and ease of maintenance goals.

In conclusion, we show that the optimization problem formulated in Section 7 recommends a different architecture for each scenario. Intuitively, the recommendations are built upon the requirements formulated in Section 4 which — although reasonable due to having been elicited from the literature — may not be representative of all eHealth, intelligent transportation systems, or smart building applications. In fact, our objective is not to propose the architectures previously identified for real-world deployments of CAC schemes, but instead to prove the effectiveness of the optimization problem and the flexibility in modeling different scenarios. Indeed, organizations can use our optimization problem as a starting point, adding further relevant goals and fine-tuning the influences of entity-domain assignments on goals according to their context.

9. Experimental Evaluation

We now present a thorough performance evaluation of the CAC scheme described in Section 6 and implemented in CryptoAC (as discussed in Section 6.4). To this end, we design and conduct four different experiments (*EXP*):

- *EXP-1* - some researchers (e.g., [59, 38, 62]) argue that IoT devices may struggle to support TLS handshakes and key derivation algorithms. However, IoT devices may also struggle to obtain symmetric keys of resources in CAC. Therefore, in this experiment we compare how long it takes for an MQTT client to set up the communication with the broker when using CryptoAC and when using TLS with respect to a baseline configuration — i.e., with no security mechanism (Section 9.1);
- *EXP-2* - after having established a connection with the broker (covered in *EXP-1*), MQTT clients publish messages and subscribe to topics as expected according to the behavior of the application. This experiment concerns the overhead imposed over the communication by the chosen security mechanism — i.e., either TLS or CryptoAC — with respect to the baseline configuration. In particular, 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 TLS and CryptoAC (Section 9.2);

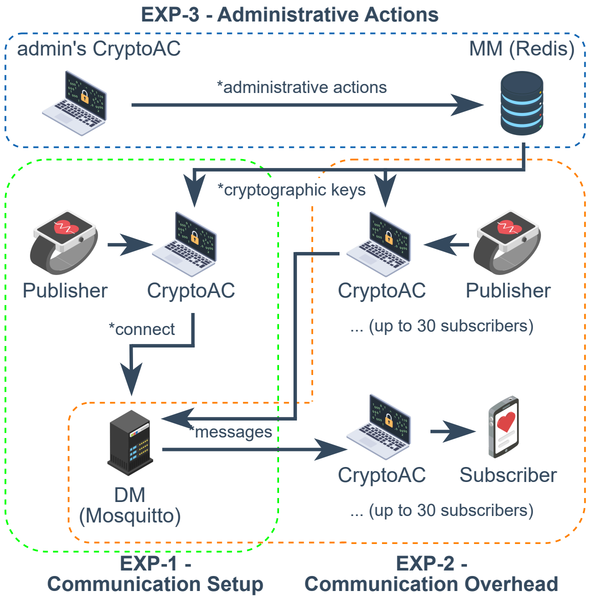

Figure 8: Setup of the Experimental Evaluation

- *EXP-3* - while MQTT clients communicate (see *EXP-2*), the administrator likely updates the CAC policy. For instance, roles may be added and permissions may be removed. In this experiment, we measure the performance of administrative actions — that is, those reported in Table 3 for which the pseudocode was provided in Figure 3 (Section 9.3);
- *EXP-4* - as explained in Sections 5.1 and 6.4, our CAC scheme considers two policy states, i.e., $\mathbf{P_t}$ and $\mathbf{P_c}$. 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 9.4).

We report a graphical representation of the setup of all entities and which are actively involved in each experiment in Figure 8. Also, 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.[43]

9.1. EXP-1 - *Communication Setup*

We want to measure the overhead on the communication setup of CryptoAC and unilateral TLSv1.3 — using a self-signed certificate and the *TLS_CHACHA20_POLY1305_SHA256* cipher, the same used in CryptoAC — with respect to a baseline 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 (e.g., MQTT clients other than Eclipse Paho that may have inherently different performance). In detail — and only for this

[43] `https://github.com/stfbk/A-Secure-and-Quality-of-Service-Aware-Solution-for-the-End-to-End-Protection-of-IoT-Applications`

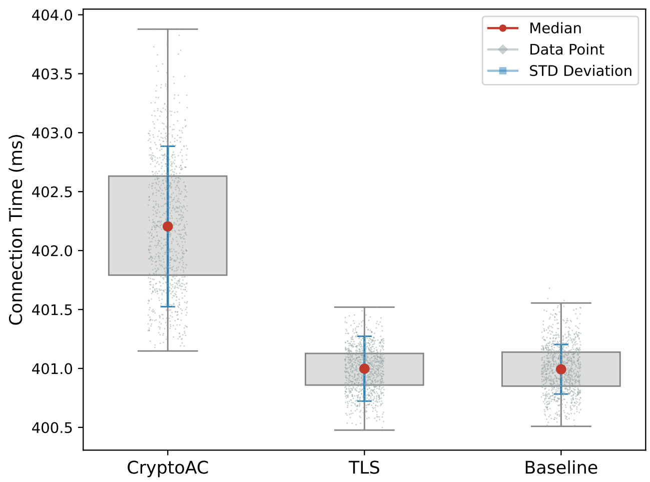

Figure 9: Communication Setup Overhead

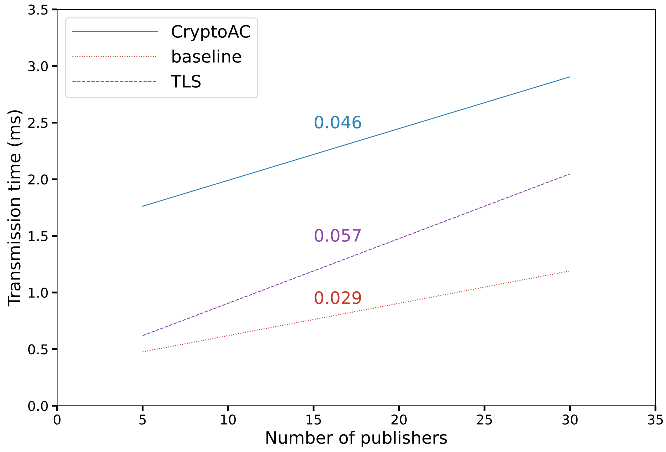

Figure 10: Communication Overhead

experiment — we remove the cryptographic computations of the CAC scheme from CryptoAC, enabling TLS in the 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). To exclude network latency — which may sensibly vary depending on the scenario — we deploy CryptoAC, the Mosquitto MQTT broker (Mosquitto v2.0.18) and the Redis datastore (v.7.0-RC2) within 3 Docker containers[44] 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 (see Figure 8 and the replication package). Specifically, we measure the difference in time before and after completing the Paho *.connect* method; we repeat the experiment 1,000 times for each configuration to mitigate measurement errors.

Results. We report the obtained results in Figure 9 as three gray box plots for CryptoAC, TLS, and the baseline configurations, respectively. The box plots are bounded by lower and upper quartiles, while we compute upper and lower whiskers as the default — i.e., 1.5× the upper and the 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). Besides, we represent the standard deviation of measures between blue whiskers. 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 TLS adds negligible overhead to the communication setup, while CryptoAC adds an overhead of 1.211ms. We believe that the better performance of TLS may be (partly) due to the fact that 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 validity of the certificate. Regardless, we believe that the overhead of CryptoAC (i.e., 1.211ms) is still acceptable in the majority of IoT 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.

9.2. EXP-2 - *Communication Overhead*

We want to evaluate the scalability of the three configurations — namely CryptoAC, 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 9.1). 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 the header of Table 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 receipt of the message by a subscriber of that topic. Intuitively, the transmission time includes the cryptographic computations of CAC and 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 num-

Table 7: Median Transmission Times (in ms) in *EXP-2*

Configuration	Number of Publishers (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

[44] https://www.docker.com/

ber of publishers and subscribers. In this regard, we expect CryptoAC to exhibit higher scalability with respect to TLS. In fact, while TLS requires the MQTT broker to de/encrypt each MQTT message for each publisher/subscriber individually using the corresponding symmetric session key, CryptoAC distributes cryptographic computations among MQTT clients.

As in *EXP-1*, to exclude network latency, we deploy a specific number of instances of CryptoAC (i.e., the publishers and the subscribers), the Mosquitto MQTT broker, and the Redis datastore within Docker containers (more precisely, a container for Mosquitto, one for Redis and more containers for publishers and subscribers - 5/5, 10/10...30/30). We deploy all containers 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.[45] We choose this device to ensure that each Docker container can run in a dedicated CPU to avoid competition over computational resources (see Figure 8 and the replication package). 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 1,000, the default threshold) via the *max_inflight_messages* and *max_queued_messages* parameters, respectively.[46] If this was not the case, messages exceeding the default thresholds would be discarded by the broker — regardless of their Quality of service (QoS) value[47] — potentially leading to biased results. Finally, to measure transmission time, we use `Locust`, an open-source load testing tool,[48] with the `ACME` extension for AC enforcement mechanisms.[49] We refer the interested reader to the replication package for more details.

The experiment thus consists in asking half of 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 QoS 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 — see EXP-1 for that), the 5 subscribers subscribe to a topic with QoS 1, and then each one of the 5 publishers sends 500 MQTT messages with QoS 1 to that topic; the total number of received MQTT messages amounts then to 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 10, together with the linear regressions (blue solid, purple dashed, and red dotted lines for CryptoAC, TLS, and baseline, respectively; slopes — or gradients — 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 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. Figure 10 shows 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). Moreover, we remark that TLS provides point-to-point data protection, while CAC provides end-to-end data protection (for a more detailed comparison between TLS and CAC, see Section 10). Finally, as reported in Section 9.1, 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 9.2. As a final remark, we note that the regression lines indicate that CryptoAC would perform even 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.

9.3. EXP-3 - *Administrative Actions*

We want to evaluate the performance of the administrative actions in our CAC scheme from three points of view: *algebraic cost*, *cryptographic time* and *administrative time*. We report the obtained results in Table 9. This experiment uses the same hardware and software configuration of EXP-1 (Ubuntu 18.04 on an Intel(R) Core(TM) i7-11370H, 16GB of RAM, CryptoAC and Redis datastore v.7.0-RC2 within Docker containers — see Figure 8 and the replication package).

Algebraic Cost. This is the number and type of cryptographic computations involved in each administrative action. We extract the algebraic costs from Figure 3, considering digital signature computations as well — i.e., a create signature computation $\mathbf{Sign^{Sig}}$ for each element added to $\mathbf{P_c}$ and a verify signature computation $\mathbf{Ver^{Sig}}$ 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., $\mathbf{Gen^{Pub}}$ represents the cost of generating an asymmetric encryption key pair). We use the symbol $|\cdot|$ to indicate the cardinality of a set (i.e., the number of its elements).

Table 8: Execution Time of Cryptographic Computations

Computation	Time	Computation	Time
$\mathbf{Gen^{Sig}}$ (EdDSA-Ed25519)	0.031ms	$\mathbf{Gen^{Pub}}$ (ECDH-X25519)	0.031ms
$\mathbf{Gen^{Sym}}$ (XChaCha20-Poly1305)	0.003ms	$\mathbf{Enc^{S}}$ (XChaCha20-Poly1305)	0.156ms/KB
$\mathbf{Dec^{S}}$ (XChaCha20-Poly1305)	0.176ms/KB	$\mathbf{Enc^{P}}$ (ECDH-X25519)	0.098ms
$\mathbf{Dec^{P}}$ (ECDH-X25519)	0.070ms	$\mathbf{Ver^{Sig}}$ (EdDSA-Ed25519)	0.090ms
$\mathbf{Sign^{Sig}}$ (EdDSA-Ed25519)	0.047ms		

[45] https://aws.amazon.com/ec2/instance-types/c6i/
[46] https://mosquitto.org/man/mosquitto-conf-5.html
[47] https://mosquitto.org/man/mqtt-7.html
[48] https://locust.io/
[49] https://github.com/stfbk/ACME

Table 9: Administrative Actions Performance Evaluation (in milliseconds)

Action	Algebraic Cost	Cryptographic Time	Administrative Time
$\underline{init_A()}$	$\mathbf{Gen^{Pub}} + \mathbf{Gen^{Sig}} + \mathbf{Enc^{P}} + (3 \cdot \mathbf{Sign^{Sig}})$	0.301ms	0.773ms
$\underline{addUser_A(u)}$	$\mathbf{Sign^{Sig}}$	0.047ms	0.759ms
$\underline{deleteUser_A(u)}$	$(\mathbf{Ver^{Sig}} + \underline{revokeUserFromRole_A(u,r)}) \cdot \lvert\{r : (u,r,-,-) \in \mathbf{UR_c}\}\rvert$		
$\underline{addRole_A(r)}$	$\mathbf{Gen^{Pub}} + \mathbf{Gen^{Sig}} + \mathbf{Enc^{P}} + (2 \cdot \mathbf{Sign^{Sig}})$	0.254ms	1.359ms
$\underline{deleteRole_A(r)}$	$(\mathbf{Ver^{Sig}} + \underline{revokePermissionFromRole_A(r, \langle f, \mathbf{OP}\rangle)}) \cdot \lvert\{f : (r,f,-,-,-,-,-,-) \in \mathbf{PA_c}\}\rvert$		
$\underline{addResource_A(f, fc, \mathsf{Enf_c})}$	$(2 \cdot \mathbf{Sign^{Sig}}) + \mathbf{Gen^{Sym}} + \mathbf{Enc^{P}} + \mathbf{Enc^{S}}$	0.195ms + 0.156ms/KB	1.592ms + 0.156ms/KB
$\underline{deleteResource_A(f)}$	No cryptographic computation involved		1.622ms
$\underline{assignU_A(u,r)}$	$\mathbf{Dec^{P}} + \mathbf{Enc^{P}} + \mathbf{Sign^{Sig}} + \mathbf{Ver^{Sig}}$	0.305ms	4.391ms
$\underline{assignPermissionToRole_A(r, \langle f, op\rangle)}$	If $(r,f,-,-,-,-,-,-) \notin \mathbf{PA_c}$, then $((3 \cdot \mathbf{Ver^{Sig}}) + \mathbf{Sign^{Sig}} + \mathbf{Dec^{P}} + \mathbf{Enc^{P}})$, else $(\mathbf{Ver^{Sig}} + \mathbf{Sign^{Sig}})$	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
$\underline{revokeUserFromRole_A(u,r)}$	$\mathbf{Gen^{Pub}} + \mathbf{Gen^{Sig}} + \mathbf{Dec^{P}} + (2 \cdot \mathbf{Ver^{Sig}}) + \mathbf{Sign^{Sig}} + (\mathbf{Ver^{Sig}} + \mathbf{Sign^{Sig}} + \mathbf{Enc^{P}}) \cdot \lvert\{u' : (u',r,-,-) \in \mathbf{UR_c} \wedge u' \neq u\}\rvert + \lvert\{f : (r,f,-,-,-,-,-,-) \in \mathbf{PA_c}\}\rvert \cdot (2 \cdot (\mathbf{Ver^{Sig}} + \mathbf{Sign^{Sig}}) + \mathbf{Gen^{Sym}} + \mathbf{Enc^{P}} + (2 \cdot \mathbf{Ver^{Sig}} + \mathbf{Sign^{Sig}} + \mathbf{Enc^{P}}) \cdot (\lvert\{(r',f') : (r',f',-,-,-,-,-,-) \in \mathbf{PA_c} \wedge r' \neq r\}\rvert))$	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) $\cdot(\lvert\{(r',f') : (r',f',-,-,-,-,-,-) \in \mathbf{PA_c} \wedge r' \neq r\}\rvert))$
$\underline{revokePermissionFromRole_A(r, \langle f, op\rangle)}$	$\mathbf{Ver^{Sig}}$ + (if $(r,f,-,-,-,-,-,op'') \in \mathbf{PA_c} \wedge op'' \subseteq op$, then $(\mathbf{Ver^{Sig}} + \mathbf{Gen^{Sym}} + \mathbf{Sign^{Sig}} + (2 \cdot \mathbf{Ver^{Sig}} + \mathbf{Sign^{Sig}} + \mathbf{Enc^{P}}) \cdot \lvert\{r' : (r',f,-,-,-,-,-,-) \in \mathbf{PA_c} \wedge r' \neq r\}\rvert)$, else if $op \cap op'' \neq \emptyset$ then $\mathbf{Sign^{Sig}}$)	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 time obtained by summing the execution time — when run in CryptoAC — of each cryptographic computation involved in each administrative action. As said in Section 6.4, we choose the Sodium[50] cryptographic library that uses ECDH-X25519 for asymmetric en/decryption, EdDSA-Ed25519 for asymmetric signature creation/verification and XChaCha20-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 9.4) — 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 $\mathbf{Sign^{Sig}}$ and $\mathbf{Ver^{Sig}}$ — are 460 bytes (i.e., the biggest block of data that is produced in $\mathbf{P_c}$, as later discussed in Section 9.4 and shown in Table 10).

Administrative Time. This is the time of each administrative action 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 Redis datastore). We employ `kotlinx.benchmark`[51] as a library for running benchmarks, where each benchmark consists in executing an administrative action.

[50] `https://libsodium.gitbook.io/doc/`

[51] `https://github.com/Kotlin/kotlinx-benchmark`

Results. We report the obtained results in Table 9. From the table, we can see that the administrative time of all administrative actions is reasonably bounded to a few milliseconds (from 0.759ms for $\underline{addUser_A(u)}$ to 5.312ms for $\underline{assignPermissionToRole_A(r, \langle f, op\rangle)}$). However, we can also see that the performance of 4 administrative actions depends on the state of the AC policy (e.g., on the number of role-permission assignments). We explore this aspect more in detail, focusing on $\underline{revokeUserFromRole_A(u,r)}$ (Section 9.3.1) and $\underline{revokePermissionFromRole_A(r, \langle f, op\rangle)}$ (Section 9.3.2), since the performance of $\underline{deleteUser_A(u)}$ and $\underline{deleteRole_A(r)}$ can be then derived easily.

9.3.1. 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., $\lvert\{u' : (u',r,-,-) \in \mathbf{UR_c} \wedge u' \neq u\}\rvert$), (*ii*) the number of resources for which a new symmetric key has to be distributed (i.e., $\lvert\{f : (r,f,-,-,-,-,-,-) \in \mathbf{PA_c}\}\rvert$) and (*iii*) the number of other roles to which the new symmetric keys have to be distributed (i.e., $\lvert\{(r',f') : (r',f',-,-,-,-,-,-) \in \mathbf{PA_c} \wedge r' \neq r\}\rvert$). Hence, we measure the performance of the action $\underline{revokeUserFromRole_A(u,r)}$ by varying these three influencing parameters — one at a time — from 1 to 500 by a step of 5. We report the results in Section 9.3.1 together with the linear regressions (blue solid lines, slopes — or gradients — 1.071, 3.827 and 0.997, respectively). Moreover, we also re-

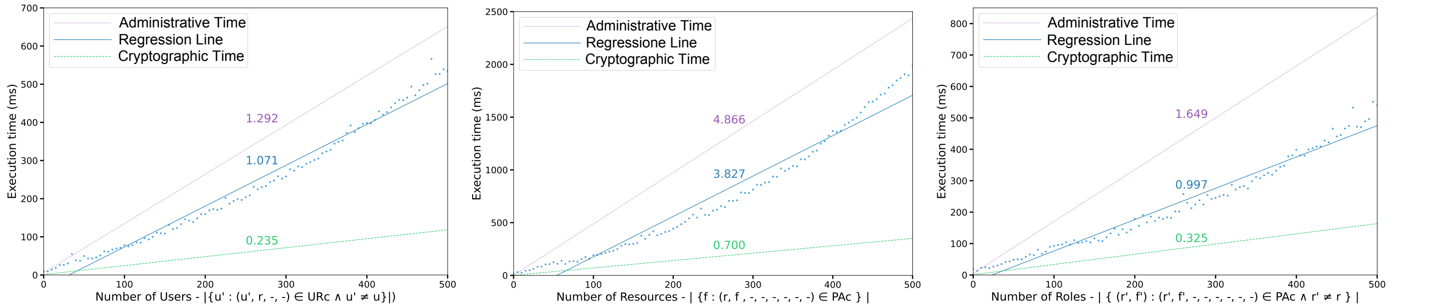

Figure 11: Performance of $\underline{revokeUserFromRole_A(u, r)}$

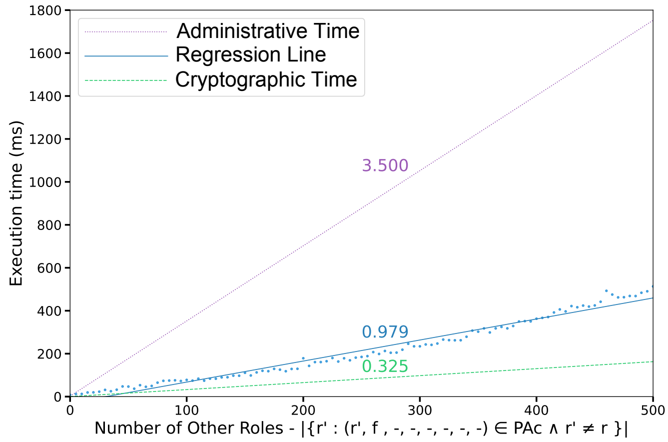

Figure 12: Performance of $\underline{revokePermissionFromRole_A(r, \langle f, op\rangle)}$

port the linear regressions on the "Cryptographic Time" (green dashed lines, slopes 0.235, 0.700, and 0.325, respectively) and the "Administrative Time" (purple dotted lines, slopes 1.292, 4.886, and 1.649, respectively) that we compute by simply multiplying the corresponding values in Table 9 for each point on the X axes (i.e., from 0 to 500 by a step of 5).

9.3.2. 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 $\underline{revokePermissionFromRole_A(r, \langle f, op\rangle)}$ by varying this influencing parameter from 0 to 500 by a step of 5. We report the results in Figure 12 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 "Administrative Time" (purple dotted line, slope 0.325) that we compute by simply multiplying the corresponding values in Table 9 for each point on the X axis (i.e., from 0 to 500 by a step of 5).

Discussion. Figures 11 and 12 show that the cryptographic and the administrative time are, respectively, a lower and an upper bound to the actual execution time. These results are reasonable, as administrative actions in CryptoAC have — in addition to cryptographic computations — computational costs (e.g., connection toward the Redis datastore) 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 $\underline{revokePermissionFromRole_A(r, \langle f, op\rangle)}$ increases by 0.979ms (see Figure 12) for each role to which f's new key has to be distributed. Overall, the slopes suggest good scalability, especially when considering that the influencing parameters of $\underline{revokeUserFromRole_A(u, r)}$ and $\underline{revokePermissionFromRole_A(r, \langle f, op\rangle)}$ concern the number of users, roles and resources involved in these actions, and not the total number of users, roles and resources in $\mathbf{P_c}$ — which may be higher. Finally, we highlight that administrative actions can be executed in more powerful devices than IoT, and that they can be scheduled for when the application is usually idling or unloaded (e.g., during the night).

9.4. EXP-4 - Policy States Size

The size of $\mathbf{P_t}$ and $\mathbf{P_c}$ in our CAC scheme can be easily determined from the corresponding definitions (Table 4) and the implementation details of CryptoAC such as the cryptographic algorithms employed and the configuration of Redis (see Section 6.4), thus without running an actual experiment. We manually measure the size of other elements of $\mathbf{P_c}$ — e.g., $\mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}^{enc}_u}\left(\mathbf{k}^{enc}_{(r,v_r)}\mathbf{k}^{dec}_{(r,v_r)}, \mathbf{k}^{ver}_{(r,v_r)}, \mathbf{k}^{sig}_{(r,v_r)}\right)$ — by running CryptoAC.

Results. We report the size of each set of $\mathbf{P_t}$ and $\mathbf{P_c}$ in Table 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. [31] from real-world datasets, which we report in Table 11 along with the corresponding size of $\mathbf{P_t}$ and $\mathbf{P_c}$ (computed by simply multiplying the numbers in Tables 10 and 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 2,443KB 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.

10. Security Comparison with TLS

We now discuss and compare the security properties offered by our CAC scheme (presented in Section 6) with respect to

Table 10: Size of Metadata

$\mathbf{P_t}$ Symbol	$\mathbf{P_t}$ Size (in bytes)	$\mathbf{P_c}$ Symbol	$\mathbf{P_c}$ Size (in bytes)
$u \in \mathbf{U_t}$	50	$(u, p_u, \mathbf{k}_u^{\mathbf{enc}}, \mathbf{k}_u^{\mathbf{ver}}) \in \mathbf{U_c}$	(50, 50, 32, 32) = 164
$r \in \mathbf{R_t}$	50	$(r, p_{(r,v_r)}, \mathbf{k}_{(r,v_r)}^{\mathbf{enc}}, \mathbf{k}_{(r,v_r)}^{\mathbf{ver}}, v_r) \in \mathbf{R_c}$	(50, 50, 32, 32, 8) = 172
$f \in \mathbf{F_t}$	50	$(f, p_{(f,v_f)}, v_f, Enf) \in \mathbf{F_c}$	(50, 50, 8, 1) = 109
$(u, r) \in \mathbf{UR_t}$	(50, 50) = 100	$(u, r, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}_u^{\mathbf{enc}}}\left(\mathbf{k}_{(r,v_r)}^{\mathbf{enc}} \mathbf{k}_{(r,v_r)}^{\mathbf{dec}}, \mathbf{k}_{(r,v_r)}^{\mathbf{ver}}, \mathbf{k}_{(r,v_r)}^{\mathbf{sig}}\right), v_r) \in \mathbf{UR_c}$	(50, 50, 352, 8) = 460
$(r, \langle f, op \rangle) \in \mathbf{PA_t}$	(50, 50, 1) = 101	$(r, f, p_{(r,v_r)}, p_{(f,v_f)}, \mathbf{Enc}^{\mathbf{P}}_{\mathbf{k}_{(r,v_r)}^{\mathbf{enc}}}\left(\mathbf{k}_{(f,v_f)}^{\mathbf{sym}}\right), v_f, v_r, op) \in \mathbf{PA_c}$	(50, 50, 50, 50, 80, 8, 8, 1) = 297

TLS. We report the comparison in Table 12 and explain each property below.

Confidentiality. The property for which a given message can be read only by the intended recipient(s). As shown in Section 6.3, CAC provides end-to-end confidentiality. In fact, in CAC a message can be read — that is, decrypted — by the subscribers of the corresponding topic only. Conversely, TLS offers hop-to-hop confidentiality only, thus mediating agents (e.g., the Edge provider operating the broker) can easily read messages.

Integrity. The property for which any alteration of a message can be noticed by the intended recipient(s). Both our CAC scheme and TLS provide integrity by using Message Authentication Codes (MACs) in AEAD (see Section 6.4 and [10]). However, while CAC provides end-to-end integrity, TLS offers hop-to-hop integrity only. In other words, when using TLS, mediating agents could potentially modify messages without the clients 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 by which the compromise of a client's private key — where the compromise happens in the future — does not compromise the confidentiality of messages exchanged by that client in the past. The latter is the property for which — at the network level — past messages cannot be replayed by an attacker in such a way to be accepted by (honest) clients. While the new version of TLS (v1.3) provides forward secrecy by default thanks to ephemeral key pairs in the Diffie-Hellman key agreement protocol (see [56]), our CAC scheme 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) messages to which u had access at the time. Nonetheless, we note that the value of information for data exchanged in IoT applications typically decreases over time [6], hence reducing the impact of a successful attack. Furthermore, the impact of an attacker — intended as the number of messages the attacker can read after having compromised a user's private key — can be reduced by periodically rotating users' (and roles') private keys. Regarding replay attack protection (at the network level), both our CAC scheme and TLS provide this property thanks to the use of AEAD which allows adding contextual information — such as nonces, version numbers, and timestamps [55] — when creating a ciphertext (see Section 6.4).

Table 11: RBAC Policy States from [31] and Policy Size

State	Set $\lvert U \rvert$	$\lvert R \rvert$	$\lvert F \rvert$	$\lvert UR \rvert$	$\lvert PA \rvert$	Size (in KB) $\mathbf{P_t}$	$\mathbf{P_c}$
domino	79	20	231	75	629	85	257
emea	35	34	3,046	35	7,211	867	2,443
firewall	365	60	709	1,130	3,455	506	1,654
firewall	325	10	590	325	1,136	189	592
healthc	46	13	46	55	359	46	143
univers	493	16	56	495	202	96	369

Table 12: Comparison on Security Properties

Security Property		CAC	TLS
Confidentiality	Point-to-Point	✓	✓
	End-to-end	✓	✗
Integrity	Point-to-Point	✓	✓
	End-to-end	✓	✗
Network-Level	Forward Secrecy	✗	✓
	Replay Attack Protection	✓	✓
Application-Level	Replay Attack Protection	✓	n/a
	Authorization	✓	n/a

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. The latter is the property for which — at the application level — past messages cannot be replayed by an attacker in such a way to be accepted by (honest) clients. 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 property, we note that, when using TLS, mediating agents could potentially replay messages without the clients noticing. Being TLS a security mechanism at the network level, we mark these two properties as not applicable (n/a) to TLS in Table 12.

From Table 12 it is easy to see that our CAC scheme, although providing several properties, is not a "silver bullet" security mechanism. Instead, its adoption should be evaluated on a case-by-case basis according to the underlying scenario (e.g., if forward secrecy is a stringent requirement, then our CAC scheme alone would not be an adequate solution) More importantly — as also mentioned in Section 1 — we remark that our CAC scheme can be used in synergy with other security mechanisms, so to achieve a more complete and throughout protection of sensitive data.

11. Discussion

We now elaborate on the implications as well as the limitations of our work in the broader context of data protection in IoT applications.

Implications. As already generally considered by other researchers (see similarities and differences in Section 2 and Table 2), our CAC scheme combines two separate security aspects: AC policy enforcement and end-to-end protection for data in transit. This combination provides a foundation for research on broader, lightweight solutions that, however, not only protect data from unauthorized access but also do so in a manner that is aware of quality of service. In fact, a key contribution of our work is (the extension of) the optimization methodology in [12] that accounts for objectives like scalability and reliability while also managing risk from likely attackers. In fact, we argue that security is not absolute, and its achievement must instead be balanced through trade-off analysis with that of performance goals and trust assumptions relevant to the underlying scenario [11]. In other words, our work underscores the necessity for nuanced security mechanisms that are mindful of quality of service. Furthermore, we show how the same security mechanism may be flexibly deployed in multiple IoT applications — namely, Remote Patient Monitoring (Section 4.1), Cooperative Maneuvering (Section 4.2), and Smart Lock (Section 4.3) — each with its own performance goals and trust assumptions. The use of the same or similar security mechanisms — although differently configured — may enable or at least facilitate the integration of IoT applications deployed in different scenarios, enabling the development of innovative yet secure services. Finally, our CAC scheme is designed for topic-based publish-subscribe protocols such as the widely used MQTT. In fact, grounding novel security mechanisms to real-world technologies allows for reducing integration overhead, leading to practical deployments and accelerating their adoption and potential impact. Summarizing, we hope our work will stimulate research on comprehensive (i.e., combining multiple security aspects), quality of service-aware, cross-scenario, and readily deployable security mechanisms for data protection in IoT applications.

Limitations. Intuitively, our work also presents some limitations. First, as discussed in Section 10, our CAC scheme does not inherently provide forward secrecy; compromise of users private keys at a future time could enable an attacker to decrypt past messages — if those messages and all corresponding keys had been cached. While forward secrecy could be achieved in part by periodically rotating keys or by blending in more sophisticated cryptographic techniques (e.g., employing ephemeral keys), these additions would introduce further complexity and computational overhead. Then, the chosen RBAC model may be sufficient for many IoT applications but is not fine-grained and does not capture dynamic and contextual conditions. To this end, we suggest integrating traditional AC with CAC, at the cost of addressing collusions. More advanced context- or attribute-based models (e.g., ABAC) are feasible to incorporate but may require re-engineering policy representation or a higher computational overhead. Moreover, certain administrative actions — particularly, revocation of permissions — entail key rotations which may become intensive in large-scale IoT applications. Although our experiments show that revocation remains generally bounded to a few milliseconds (see Section 9.3), a surge of simultaneous revocation actions may increase execution time or lead to synchronization challenges when real-time or near-real-time reactions are required. Finally, as we discuss in Section 6.4, CryptoAC is not yet production-ready and several other implementation-level aspects (e.g., robust key management, continuous vulnerability monitoring, auditing compliance) would need to be considered.

12. Conclusion

In this paper, we addressed the problem of quality of service-aware end-to-end protection of IoT communications in presence of external attackers, malicious insiders, and partially trusted agents. First, we introduced a CAC scheme for enforcing RBAC policies. Then, we formalized an optimization problem to fine-tune the deployment of the entities composing our CAC scheme to find the best trade-off between security and quality of service. To demonstrate the benefits of our contributions, we discussed the use of our CAC scheme and the application of our optimization problem in 3 different scenarios for IoT applications (i.e., Remote Patient Monitoring, Cooperative Maneuvering, and Smart Lock). In this regard, we highlight that our CAC scheme can ideally be applied to any application employing topic-based publish-subscribe protocols — not limited to IoT but also including, e.g., cloud native applications [16]. Finally, we implemented our CAC scheme into an open-source tool named CryptoAC[52] and conducted a thorough performance evaluation considering 4 different experiments. The results show that CryptoAC offers greater scalability than TLS, although entailing a non-negligible administrative overhead (see Section 9.3). Besides performance, we also compared the security properties offered by our CAC scheme with respect to TLS. Despite its limitations (see Section 11), our CAC scheme and its implementation in CryptoAC address an important niche in IoT communications security, offering a unified way to cryptographically protect data and enforce AC directly at the application level. In addition, the formalization of the optimization problem enables a rigorous yet flexible trade-off analysis, allowing for evaluating the adoption (and configuration) of our contributions on a case-by-case basis according to the performance and security requirements of the underlying scenario.

[52] `https://cryptoac.readthedocs.io/`

Future Work. We can identify a number of compelling research directions stemming from our work. First, we are working to extend our CAC scheme with ABE to allow for enforcing more expressive and fine-grained ABAC policies. In particular, we plan to refer to the well-established NIST SP 800-162 [40] as a starting point to identify an ABAC model describing the complete set of available actions for ABAC (as done for core RBAC in Table 3). Concerning the type of the underlying ABE cryptosystem, we are considering CP-ABE over KP-ABE since, as argued in [66], CP-ABE is more suitable than KP-ABE for enforcing AC policies. Moreover, we plan to investigate the use of sticky policies, similarly to the work in [61]. This research direction would also allow us to investigate the integration of CAC with context-based triggers (e.g., date and time) for contextually-aware AC policy enforcement as done in, e.g., [24]. Again, we highlight that the combination of CAC with traditional (e.g., centralized) AC enforcement mechanisms comes at the cost of addressing possible collusions between users and the (agents managing the) traditional enforcement mechanisms. At the same time, we are designing a CAC scheme that can be applied to data both in transit and at rest — hence, aiming to ensure comprehensive protection for data. Besides, we are investigating the use of TEEs for guaranteeing confidentiality and integrity in IoT scenarios, as in [60], and to provide different levels of security, as in [47]. We also plan to further develop the optimization problem discussed in Section 7 by implementing it into an algorithm to be integrated with orchestrators (e.g., Kubernetes[53]) and other open source technologies (e.g., FogAtlas[54]) for managing Cloud-native applications. Moreover, as said at the end of Section 9.2, CryptoAC may perform better than TLS — in terms of absolute communication overhead — when considering 110+ publishers; hence, we plan to conduct further experiments with a higher number of publishers and subscribers to verify such hypothesis. Finally, we aim to consider additional metrics in the experimental evaluation (e.g., energy efficiency, computational overhead) to provide a more comprehensive understanding of the performance of our CAC scheme and its implementation in CryptoAC.

Acknowledgments

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

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

[2] Tahir Ahmad, Umberto Morelli, and Silvio Ranise. Deploying access control enforcement for IoT in the cloud-edge continuum with the help of the CAP theorem. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*, pages 213–220. ACM, 2020.

[3] Tahir Ahmad, Umberto Morelli, Silvio Ranise, and Nicola Zannone. A lazy approach to access control as a service (acaas) for iot: An aws case study. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, SACMAT '18, page 235–246, New York, NY, USA, 2018. Association for Computing Machinery.

[4] Tahir Ahmad, Umberto Morelli, Silvio Ranise, and Nicola 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.

[5] P. S. Akshatha and S. M. Dilip Kumar. Enhancing security mechanism of MQTT protocol using payload encryption. In Ram Sarkar, Sujata Pal, Subhadip Basu, Dariusz Plewczynski, and Debotosh Bhattacharjee, editors, *Proceedings of International Conference on Frontiers in Computing and Systems*, volume 690, pages 199–208. Springer Nature Singapore, 2023. Series Title: Lecture Notes in Networks and Systems.

[6] Faiga Alawad and Frank Alexander Kraemer. Value of information in wireless sensor network applications and the iot: A review. *IEEE Sensors Journal*, 22(10):9228–9245, 2022.

[7] Alessandro Armando, Matteo Grasso, Sander Oudkerk, Silvio Ranise, and Konrad Wrona. Content-based information protection and release in NATO operations. In *Proceedings of the 18th ACM symposium on Access control models and technologies - SACMAT '13*, page 261. ACM Press, 2013.

[8] Alessandro Armando, Sander Oudkerk, Silvio Ranise, and Konrad Wrona. Formal modelling of content-based protection and release for access control in NATO operations. In Jean Luc Danger, Mourad Debbabi, Jean-Yves Marion, Joaquin Garcia-Alfaro, and Nur Zincir Heywood, editors, *Foundations and Practice of Security*, volume 8352, pages 227–244. Springer International Publishing, 2014. Series Title: Lecture Notes in Computer Science.

[9] Elaine Barker. Recommendation for key management: part 1 - general, 2020.

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

[11] Stefano Berlato. *A Security Service for Performance-Aware End-to-End Protection of Sensitive Data in Cloud Native Applications*. Phd thesis, University of Gemoa, May 2024. Available at `https://hdl.handle.net/11567/1174596`.

[12] Stefano Berlato, Roberto Carbone, Adam J. Lee, and Silvio 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.

[13] Stefano Berlato, Marco Centenaro, and Silvio Ranise. Smart card-based identity management protocols for v2v and v2i communications in ccam: A systematic literature review. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–18, 2021.

[14] Stefano Berlato, Silvio Cretti, Domenico Siracusa, and Silvio Ranise. Multi-objective microservice orchestration: Balancing security and performance in CCAM. In *2024 27th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, pages 88–90. IEEE, 2024.

[15] Stefano Berlato, Umberto Morelli, Roberto Carbone, and Silvio Ranise. End-to-end protection of IoT communications through cryptographic enforcement of access control policies. In Shamik Sural and Haibing 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.

[16] Stefano Berlato, Matteo Rizzi, Matteo Franzil, Silvio Cretti, Pietro De Matteis, and Roberto Carbone. Work-in-progress: A sidecar proxy for usable and performance-adaptable end-to-end protection of communications in cloud native applications. In *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 706–711. IEEE, 2024.

[17] Joseph Bonneau and Ilya Mironov. Cache-collision timing attacks against AES. In Louis Goubin and Mitsuru 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.

[18] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh 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.

[53] `https://kubernetes.io/`

[54] `https://fogatlas.fbk.eu/`

ACM, 2012.

[19] Francesco Buccafurri, Vincenzo De Angelis, and Sara Lazzaro. MQTT-i: Achieving end-to-end data flow integrity in MQTT. *IEEE Transactions on Dependable and Secure Computing*, 21(5):4717–4734, 2025.

[20] Marco Calabretta, Riccardo Pecori, and Luca Veltri. A token-based protocol for securing MQTT communications. In *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6. IEEE, 2018.

[21] Marco Centenaro, Stefano Berlato, Roberto Carbone, Gianfranco Burzio, Giuseppe Faranda Cordella, Roberto Riggio, and Silvio Ranise. Safety-related cooperative, connected, and automated mobility services: Interplay between functional and security requirements. *IEEE Vehicular Technology Magazine*, 16(4):78–88, 2021.

[22] Marco Centenaro, Stefano Berlato, Roberto Carbone, Gianfranco Burzio, Giuseppe Faranda Cordella, Roberto Riggio, and Silvio Ranise. Safety-related cooperative, connected, and automated mobility services: Interplay between functional and security requirements. *IEEE Vehicular Technology Magazine*, pages 2–12, 2021.

[23] Lily Chen, Dustin Moody, Andrew Regenscheid, Angela Robinson, and Karen Randall. Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters.

[24] Pietro Colombo and Elena Ferrari. Access control enforcement within MQTT-based internet of things ecosystems. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, pages 223–234. ACM, 2018.

[25] Chandramohan Dhasaratha, Mohammad Kamrul Hasan, Shayla Islam, Shailesh Khapre, Salwani Abdullah, Taher M. Ghazal, Ahmed Ibrahim Alzahrani, Nasser Alalwan, Nguyen Vo, and Md Akhtaruzzaman. Data privacy model using blockchain reinforcement federated learning approach for scalable internet of medical things. *CAAI Transactions on Intelligence Technology*, page cit2.12287, 2024.

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

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

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

[29] Bhaskara S. Egala, Ashok K. Pradhan, Venkataramana Badarla, and Saraju 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.

[30] Eman Elemam, Ayman M. Bahaa-Eldin, Nabil H. Shaker, and Mohamed A. Sobh. A secure MQTT protocol, telemedicine IoT case study. In *2019 14th International Conference on Computer Engineering and Systems (ICCES)*, pages 99–105. IEEE, 2019.

[31] Alina Ene, William Horne, Nikola Milosavljevic, Prasad Rao, Robert Schreiber, and Robert 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.

[32] Chun-I Fan, Cheng-Han Shie, Yi-Fan Tseng, and Hui-Chun Huang. An efficient data protection scheme based on hierarchical ID-based encryption for MQTT. *ACM Transactions on Sensor Networks*, 19(3):1–21, 2023.

[33] American National Standard for Information Technology. Role based access control. Standard, American National Standards Institute, Inc., February 2004.

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

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

[36] Sanjay Goel and Vicki Chen. Information security risk analysis – a matrix-based approach. `https://www.albany.edu/~goel/publications/goelchen2005.pdf`, 2005. Accessed: 2021-09-08.

[37] Sandeep Gupta, Tommaso Sacchetti, and Bruno Crispo. End-to-end encryption for securing communications in industry 4.0. In *2022 4th IEEE Middle East and North Africa COMMunications Conference (MENACOMM)*, pages 153–158. IEEE, 2022.

[38] Tobias Heer, Oscar Garcia-Morchon, René Hummen, Sye Loong Keoh, Sandeep S. Kumar, and Klaus Wehrle. Security challenges in the IP-based internet of things. *Wireless Personal Communications*, 61(3):527–542, 2011.

[39] Ahmed J. Hintaw, Selvakumar Manickam, Shankar Karuppayah, Mohammad Adnan Aladaileh, Mohammed Faiz Aboalmaaly, and Shams Ul Arfeen Laghari. A robust security scheme based on enhanced symmetric algorithm for MQTT in the internet of things. *IEEE Access*, 11:43019–43040, 2023.

[40] Vincent C. Hu, David F. Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to Attribute Based Access Control (ABAC) Definition and Considerations, 2014.

[41] Ingrid Huso, Daniele Sparapano, Giuseppe Piro, and Gennaro Boggia. Privacy-preserving data dissemination scheme based on searchable encryption, publish–subscribe model, and edge computing. *Computer Communications*, 203:262–275, 2023.

[42] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

[43] Md. Tanzim Khorshed, A.B.M. Shawkat Ali, and Saleh A. Wasimi. A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing. *Future Generation Computer Systems*, 28(6):833–851, 2012. Including Special sections SS: Volunteer Computing and Desktop Grids and SS: Mobile Ubiquitous Computing.

[44] Kathrin Klamroth. Discrete multiobjective optimization. In Matthias Ehrgott, Carlos M. Fonseca, Xavier Gandibleux, Jin-Kao Hao, and Marc Sevaux, editors, *Evolutionary Multi-Criterion Optimization*, volume 5467, pages 4–4. Springer Berlin Heidelberg, 2009. Series Title: Lecture Notes in Computer Science.

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

[46] Adam Langley, Mike Hamburg, and Sean Turner. Elliptic Curves for Security, January 2016.

[47] Lukas Malina, Gautam Srivastava, Petr Dzurenda, Jan Hajny, and Radek Fujdiak. A secure publish/subscribe protocol for internet of things. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pages 1–10. ACM, 2019.

[48] Koosha Mohammad Hossein, Mohammad Esmaeil Esmaeili, Tooska Dargahi, Ahmad Khonsari, and Mauro Conti. BCHealth: A novel blockchain-based privacy-preserving architecture for IoT healthcare applications. *Computer Communications*, 180:31–47, 2021.

[49] Muath A. Obaidat, Suhaib Obeidat, Jennifer Holst, Abdullah Al Hayajneh, and Joseph Brown. A comprehensive and systematic survey on the internet of things: Security and privacy challenges, security frameworks, enabling technologies, threats, vulnerabilities and countermeasures. *Computers*, 9(2):44, 2020.

[50] Maire O'Neill. Insecurity by design: Today's IoT device security problem. *Engineering*, 2(1):48–49, 2016.

[51] Andrea Palmieri, Paolo Prem, Silvio Ranise, Umberto Morelli, and Tahir Ahmad. Mqttsa: A tool for automatically assisting the secure deployments of mqtt brokers. In *2019 IEEE World Congress on Services (SERVICES)*, volume 2642-939X, pages 47–53, 2019.

[52] Abdullah Al-Noman Patwary, Anmin Fu, Ranesh Kumar Naha, Sudheer Kumar Battula, Saurabh Kumar Garg, Md Anwarul Kaium Patwary, and Erfan Aghasian. Authentication, access control, privacy, threats and trust management towards securing fog computing environments: A review. *CoRR*, abs/2003.00395, 2020.

[53] Jonathan Petit, Florian Schaub, Michael Feiri, and Frank Kargl. Pseudonym schemes in vehicular networks: A survey. *IEEE Communications Surveys & Tutorials*, 17(1):228–255, 2015.

[54] Horizon 2020 5G-CARMEN Project. 5g for connected and automated road mobility in the european union - deliverable d2.1 - 5g carmen use cases and requirements. Technical report, Horizon 2020 5G-CARMEN Project, may 2019.

[55] Do Tran Quang, Nguyen Minh Quan, Do Viet Hung, Trinh Viet Dung,

Doan Minh Khanh, Nguyen Xuan Quyen, and Ta Thi Kim Hue. Identity-based authenticated encryption in internet of medical things. In *2023 8th International Scientific Conference on Applying New Technology in Green Buildings (ATiGB)*, pages 129–135. IEEE, 2023.

[56] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

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

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

[59] Eduardo Buetas Sanjuan, Ismael Abad Cardiel, Jose A. Cerrada, and Carlos Cerrada. Message queuing telemetry transport (MQTT) security: A cryptographic smart card approach. *IEEE Access*, 8:115051–115062, 2020.

[60] Carlos Segarra, Ricard Delgado-Gonzalo, and Valerio Schiavoni. MQT-TZ: Hardening IoT brokers using ARM TrustZone : (practical experience report). In *2020 International Symposium on Reliable Distributed Systems (SRDS)*, pages 256–265. IEEE, 2020.

[61] Sabrina Sicari, Alessandra Rizzardi, Gianluca Dini, Pericle Perazzo, Michele La Manna, and Alberto Coen-Porisini. Attribute-based encryption and sticky policies for data access control in a smart home scenario: a comparison on networked smart object middleware. *International Journal of Information Security*, 20(5):695–713, 2021.

[62] Meena Singh, M.A. Rajan, V.L. 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.

[63] David Strom and Jelle Frank van der Zwet. Truth and lies about latency in the cloud. *InterxionTM white paper*, 2012.

[64] Yang Tang, Patrick P. C. Lee, John C. S. Lui, and Radia Perlman. FADE: Secure overlay cloud storage with file assured deletion. In Sushil Jajodia and Jianying 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.

[65] Dirk Thatmann, Sebastian Zickau, Alexander Forster, and Axel Kupper. Applying attribute-based encryption on publish subscribe messaging patterns for the internet of things. In *2015 IEEE International Conference on Data Science and Data Intensive Systems*, pages 556–563. IEEE, 2015.

[66] Marloes Venema, Greg Alpár, and Jaap-Henk 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.

[67] Wei Yu, Fan Liang, Xiaofei He, William Grant Hatcher, Chao Lu, Jie Lin, and Xinyu Yang. A survey on the edge computing for the internet of things. *IEEE Access*, 6:6900–6919, 2018.

[68] Sherali Zeadally, Ashok Kumar Das, and Nicolas Sklavos. Cryptographic technologies and protocol standards for internet of things. *Internet of Things*, page 100075, 2019.

[69] Xingguang Zhou, Jianwei Liu, Zongyang Zhang, and Qianhong 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.