DISSERTATION / DOCTORAL THESIS

Titel der Dissertation /Title of the Doctoral Thesis

Trustworthy Context-Aware Access Control in IoT Environments Based on the Fog Computing Paradigm

verfasst von / submitted by

Nemanja Ignjatov

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der technischen Wissenschaften (Dr. techn.)

Wien, 2022 / Vienna, 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 786 880
Dissertationsgebiet lt. Studienblatt / field of study as it appears on the student record sheet:	Informatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Math. Dr.Peter Reichl, Privatdoz.

Acknowledgments

First and foremost, I would like to express my gratitude to my supervisor, Professor Peter Reichl, for all the discussions, guidance, proofreading, corrections, and kind words of encouragement and motivation over the past years.

Next, I would like to thank the Cooperative Systems Research Group for all the pleasant moments we shared in the previous years and their personal and professional support during my research at the University of Vienna. Svenja Schröder and Kaspar Lebloch deserve special gratitude for the enjoyable collaboration, numerous discussions, and constructive feedback that always led to the improvements of my thesis.

This work is supported with a netidee scholarship by the Internet Foundation Austria. For that reason, I am grateful to them for the awarded scholarship and the additional time I could invest in improving my research results.

Furthermore, I greatly thank my friends for their support and patience, especially while writing and finalizing my thesis.

Special gratitude goes to my family, especially my sister Tamara and my mother Snežana, for providing their endless support and everything they have done for me through my overall education period in the previous 25 years.

Finally and most importantly, my sincerest gratitude is dedicated to Jelena for the considerable effort she invested in proofreading this thesis in its final stages. Moreover, I am grateful for her reserveless support and love, all the moments we have shared in the previous years, and for keeping me motivated to advance each day, both professionally and personally.

netidee
FÖRDERUNGEN

Contents

Abstract

The strongly increasing connectivity and interest in today's Information Technology (IT) infrastructure resulted in a huge amount of interconnected networks building the Internet of Things (IoT). In parallel, a rapidly changing landscape of innovative services for end-users could be observed. Cloud Computing (CC) provides computing and storage resources for the vast majority of IoT services. Still, further IoT development strives for building low-latency, reliable, highly distributed systems, for which CC fails to satisfy the requirements. Computing capabilities distribution paradigms, among others Edge Computing (EC) and Fog Computing (FC), aim to deploy computing and storage resources at the IoT networks' edge. Through that, a distance gap between Things and CC Data Centers is bridged, enabling the innovative IoT services development.

Future IoT developments need to be accompanied by appropriate system architecture and security concepts to allow safe usage, overall security, and improved usability in the complex IoT ecosystem. Well-researched, traditional network security mechanisms like public-key encryption or X.509 certificates often cannot be applied in IoT systems, primarily due to the Things' computational constraints. Additionally, heterogeneity, scale, and geographical distribution of IoT networks impose challenges for the security management which traditional protocols and frameworks fail to satisfy through centralized CC architectures. Finally, security management often involves human effort for security policy configuration, which appears to be cumbersome and error-prone for non-tech-savvy users. For those reasons, improving IoT security requires rethinking of the traditional security mechanisms in multiple directions: (i) offloading and distributing security management through paradigms like EC or FC, (ii) reducing computational requirements for security procedures and protocols like encryption or TLS, and (iii) automating security policy management in IoT through data analysis in IoT environments, adapting to its current state and minimizing users' effort required for the security management.

This dissertation contributes a novel FC-based solution comprising protocols, data models, and guidelines for distributing security services at the IoT network's edge. First, trust management in the Cloud - Fog - Thing continuum is introduced, enabling scalable and reliable trust management both in local and global IoT environments. Additionally, computation requirements concerning minimization and trust management application on Things have been thoroughly examined, allowing the solution's utilization in End-to-End (E2E) security scenarios. Second, an Access Control (AC) distribution model has been developed, allowing AC operations availability in "offline" use-cases, that is, without connection to the Cloud, relying just on FC resources. Furthermore, security policy management has been automated by adapting access policy based on the IoT environment's context information. In order to develop context-aware access policies, the following contributions have been made: (1) a generic data model for integration of context information in access policies, (2) protocols for exchanging context information, and (3) guidelines for integrating context information in AC services. The designed and realized approaches have been evaluated for their performance, operability, and applicability based on a real-world Smart Home Management system - COSYLab. Lastly, the analysis of the evaluation results leads to the conclusion that proves the applicability of the implemented solution in FC-based IoT systems, ensuring beneficial capabilities for hosting distributed IoT services, such as (i) offloading computational requirements from resource-constrained Things and remote Cloud Servers and (ii) reducing overall latency of the provided IoT services.

Kurzfassung

Die Entwicklung des Internet in den letzten Jahren hin zum Internet der Dinge (Internet of Things, IoT) war durch stark wachsende Konnektivität und eine sich rasch verändernde Landschaft innovativer Dienste für den Endnutzer geprägt. Während dabei Cloud Computing die Rechen- und Speicherressourcen für die große Mehrheit der IoT-Dienste bereitstellte, zielt die weitere IoT-Entwicklung jedoch auf den Aufbau von zuverlässigen, hochgradig verteilten Systemen mit geringer Latenzzeit ab, für die herkömmliches Cloud Computing nicht ausreicht. Alternative Paradigmen zur Verteilung von Rechenkapazitäten, wie etwa Edge Computing (EC) und Fog Computing (FC), haben daher zum Ziel, Rechen- und Speicherressourcen am Rande der IoT-Netzwerke einzusetzen. Dadurch wird die Distanz zwischen den IoT Geräten und den Cloud-Computing-Rechenzentren effizient überbrückt, was die Entwicklung innovativer IoT-Dienste ermöglicht.

Allerdings müssen künftige IoT-Entwicklungen von geeigneten Systemarchitekturen und Sicherheitskonzepten begleitet werden, um eine sichere Nutzung, und verbesserte Benutzerfreundlichkeit in dem komplexen IoT-Ökosystem zu ermöglichen. Gut erforschte, traditionelle Netzwerksicherheitsmechanismen wie Public-Key-Verschlüsselung oder X.509-Zertifikate können in IoT-Systemen oft nicht angewendet werden, vor allem wegen der Rechenbeschränkungen der IoT Geräte. Darüber hinaus stellen die Heterogenität, der Umfang und die geografische Verteilung von IoT-Netzwerken neue Herausforderungen für das Sicherheitsmanagement dar, denen herkömmliche Protokolle und Rahmenwerke durch zentralisierte Cloud-Computing-Architekturen nicht gerecht werden können. Schließlich macht das Sicherheitsmanagement häufig manuelles Eingreifen bei der Konfiguration von Sicherheitsrichtlinien notwendig, was für technisch nicht versierte Benutzer umständlich und fehleranfällig ist. Aus diesen Gründen erfordert die Verbesserung von IoT-Sicherheit ein Überdenken der traditionellen Sicherheitsmechanismen in mehrere Richtungen: (i) Auslagerung und Verteilung der Sicherheitsverwaltung durch Paradigmen wie EC oder FC, (ii) Verringerung des Rechenaufwands für Sicherheitsverfahren und -protokolle wie Verschlüsselung oder Transport Layer Security, und (iii) Automatisierung der Verwaltung von Sicherheitsrichtlinien im IoT durch Datenanalyse in IoT-Umgebungen, Anpassung an den aktuellen Zustand und Minimierung des für die Sicherheitsverwaltung erforderlichen Aufwands seitens der Benutzer.

Diese Dissertation stellt eine neuartige FC-basierte Lösung vor, die Protokolle, Datenmodelle und Richtlinien für die Verteilung von Sicherheitsdiensten am Rande des IoT-Netzwerks umfasst. Zunächst wird ein neuer Ansatz zum Vertrauensmanagement im „Cloud - Fog - Thing"-Kontinuum vorgestellt, der ein skalierbares und zuverlässiges Vertrauensmanagement sowohl in lokalen als auch globalen IoT-Umgebungen ermöglicht. Sodann werden die Rechenanforderungen in Bezug auf die Minimierung und die Anwendung des Vertrauensmanagements auf IoT-Geräte untersucht, was die Nutzung der Lösung in End-to-End-Sicherheitsszenarien ermöglicht. Zudem wird ein Verteilungsmodell für die Zugangskontrolle entwickelt, das die Verfügbarkeit von Zugangskontrollvorgängen in "Offline"-Nutzungsfällen ermöglicht, d. h. ohne Verbindung zur Cloud, nur unter Nutzung von FC-Ressourcen. Darüber hinaus wird gezeigt, wie sich die Verwaltung der Sicherheitsrichtlinien automatisieren lässt, indem die Zugriffsrichtlinien auf der Grundlage der Kontextinformationen der IoT-Umgebung angepasst werden. Um kontextbezogene Zugriffsrichtlinien zu entwickeln, wurden die folgenden Beiträge geleistet: (1) ein generisches Datenmodell für die Integration von Kontextinformationen in Zugriffsrichtlinien, (2) Protokolle für den Austausch von Kontextinformationen, und (3) Richtlinien für die Integration von Kontextinformationen in Zugriffskontrolldienste. Die entworfenen und realisierten Ansätze wurden auf ihre Leistungsfähigkeit, Funktionsfähigkeit und Anwendbarkeit anhand eines realen Smart Home Management Systems evaluiert. Die Analyse der Bewertungsergebnisse beweist abschließend die Anwendbarkeit der implementierten Lösung in FC-basierten IoT-Systemen, welche vorteilhafte Fähigkeiten für das Hosting verteilter IoT-Dienste gewährleisten, wie z. B. (i) die Auslagerung von Rechenanforderungen von ressourcenbeschränkten IoT Geräten und entfernten Cloud-Servern und (ii) die Reduzierung der Gesamtlatenz der bereitgestellten IoT-Dienste.

1 Introduction

Huge advancements in the field of electronics and deployments of wireless communication systems and mobile devices in the past decade allowed the extension of everyday devices with sensing, computational, and communication capabilities and their integration into computer networks. Through that, besides interacting with the physical world, devices obtained extended communication capabilities to exchange information with each other over the Internet, which named them *Virtual Devices* or *Things*. These communication capabilities allowed remote management and control of Things and made the everyday devices more efficient, reliable, and adaptable to the environment. [Bor14; ČH18]

These advancements and introduced services have been summarized through the term *IoT* [SU22]. IoT enfolds Things with diverse characteristics like addressing and communication using Internet Protocol (IP), communication over wireless networks, integration into complex computer systems, and involvement of sensing and actuating capabilities to interact with the environment [SU22]. These characteristics enabled a set of novel services and the next wave of IT systems' technological innovations, enabling nearly limitless IoT applications deployment in various IoT application domains (e.g., Smart City, E-Health or Smart Agriculture) [Bor14; ČH18]. Irrespective of the application area, IoT applications can analyze collected information and enhance the quality of everyday life, impacting the economy and society profoundly. Moreover, increasing IoT systems' popularity leads to the integration of a vast number of Things.

1.1 Motivation & Background

IoT systems' complexity and the diversity of the applied computer science areas in IoT systems (e.g., distributed systems, data security, and machine learning) attracted the research community's attention. Moreover, enormous effort in various research, innovation, and standardization fields (e.g., network and system architectures, emerging technologies, and information security) has been invested in defining the standard tools and frameworks for the IoT development by the standardization bodies like IEEE[1], IETF[2], or NIST[3]. In 2009 European Commission launched the IoT initiative as part of the 7th Framework Program (FP7)[4] to develop architectures and optimized technologies to support heterogeneous IoT applications and services. Furthermore, the European Research Cluster on the IoT (IERC)[5] was founded to promote a common vision of IoT and interoperability between the IoT applications and the world-wide secure deployment of IoT. However, despite huge standardization efforts, no common and comprehensive framework with integrated standards under one IoT vision has been defined yet. [Bor14; ČH18; GMS15]

The initial motivation and the first phase of this dissertation occurred during my involvement in the H2020 project named *symbIoTe - a symbiosis of smart objects across IoT environments*[6]. The main goal of the symbIoTe research project was to establish an intermediary abstraction layer between various IoT environments, creating a unified view of the deployed IoT systems. Within this project, I've researched AC mechanisms and implemented the AC management solution that enables authentication and authorization mechanisms for user federation between IoT providers. For that purpose, the focus of my research and resulting contributions was on:

1 Institute of Electrical and Electronics Engineers, `https://www.ieee.org/`, last access May 2, 2022
2 Internet Engineering Task Force, `https://www.ietf.org/`, last access May 2, 2022
3 National Institute of Standards and Technology, `https://www.nist.gov/`, last access May 2, 2022
4 Seventh Framework Programme, `https://ec.europa.eu/eurostat/cros/content/fp7-projects_en`, last access May 2, 2022
5 IoT European Research Cluster, `http://www.internet-of-things-research.eu/`, last access May 2, 2022
6 symbIoTe project page, `https://www.symbiote-h2020.eu/`, last access May 2, 2022

1. The integration of the distributed, local IoT environments, published in the Deliverable 4.1[1];
2. The establishment of the authentication and authorization mechanisms between federated IoT platforms, published in the Deliverable 3.1[2];
3. The definition of the semantic data model for integration of AC schemas between IoT providers, published in the Deliverable 2.5[3].

Following the initial research contributions, the further work on this dissertation remained in the areas of network security and distributed systems, aiming to create standalone AC mechanisms that can operate in local IoT environments, capable of adapting security policies to the IoT environment's state. Based on the goals and outcomes in this dissertation's scope, I have been awarded the netIdee scholarship for the final phase of my dissertation. Using this scholarship, I further researched the aforementioned areas and developed the solution presented in this dissertation. Partial outcomes of this dissertation are provided on the scholarship's web page for my research project[7].

Inspired by the initial motivation and research efforts, this dissertation examines the following research areas:

1. (A1) System architecture concerning IoT systems' scalability, flexibility, and dynamicity;
2. (A2) Security and trust management;
3. (A3) System intelligence and self-configuration.

The goal of A1 is to define a **scalable**, **flexible**, and **efficient** architecture, able to cope with the amount of data and dynamicity of IoT systems. This architecture has to enable efficient mechanisms to retrieve all information produced by Things and provide users with access to it [Bor14]. Therefore, a hierarchical network architecture providing scalable connectivity and processing capacity based on CC is applied in IoT applications. However, CC itself is not efficient enough for all IoT applications due to the drawbacks introduced by the Cloud Servers remote deployment, such as unacceptable delay, lacking support for mobility and

1 symbIoTe Deliverable 4.1, https://zenodo.org/record/817486#.Yl1hLuhByUl, last access May 2, 2022

2 symbIoTe Deliverable 3.1, https://zenodo.org/record/817471#.Yl1in-hByUl, last access May 2, 2022

3 symbIoTe Deliverable 2.5, https://zenodo.org/record/830234#.Yl1jquhByUk, last access May 2, 2022

7 netIdee scholarship research project web page, https://www.netidee.at/trustworthy-context-aware-access-control-iot-environments-based-fog-computing-paradigm, last access May 2, 2022

location-awareness, as well as dependency on network Quality of Service (QoS) [ČH18; Sch+17; Bon+12b]. For those reasons, deployment of computing, storage and connectivity capabilities at the IoT networks' edge, using emerging paradigms like EC, Mobile Edge Computing (MCC), Mobile Cloud Computing (MEC), and FC represent attractive future research topics [ČH18; Bon+12b; RZL13]. Between these paradigms, FC stands up as a promising candidate for distribution IoT services away from CC servers. Since FC relies on the highly virtualized service deployment strategy, it increases the efficiency of the devices present in IoT environments, enabling IoT services deployment capabilities to local Fog Node (FN) devices and shifting data processing away from CC servers.

Further factors impacting IoT systems' adoption are **security and trust management** issues [BPM16; ČH18; Bor14; RZL13]. These issues may occur at various levels and affect data exchanges spanning over the ISO/OSI layers: network, transport, and application [Bor14], resulting in numerous threats for IoT systems like insecure authentication and authorization, insufficient transport encryption, insecure web interfaces and firmware deployment [ČH18]. Scope of A2 is to analyze the threats and define countermeasures using the mechanisms like **AC** and secure data transmission, which are guaranteed by using acknowledged network security principles like Confidentiality, Integrity, and Availability (CIA) [Bor14; ČH18]. However, computational, storage and communication constraints of Things introduce unique requirements for designing IoT systems security solutions, often making traditional security solutions non-practical [LC16; GMS15]. This introduces trade-offs between data security and computational- and energy-efficiency of security algorithms. Securing data transmission requires protecting communication channels using cryptographic algorithms, requiring the Things to join the IoT network securely, that is establish trust relationships and install required encryption material (keys or certificates) [Bor14]. For that purpose, a proper infrastructure for scalable and efficient **Identity Management (IdM) and key distribution** has to be deployed in IoT [RZL13; LC16; TC16]. This infrastructure also has to take into consideration the heterogeneity of Things, mostly concerning computational capabilities, which heavily affects IdM and trust management, introducing challenges in defining an universal approach for trust management in IoT [RZL13]. Furthermore, limiting access to IoT services and collected data is essential for ensuring data confidentiality and privacy, which is achieved through AC. [Bor14]. AC mechanisms involve user authentication and service access requests validation against pre-configured security policies. However, the diversity of IoT applications and the scale of connected Things requires the definition of novel AC mechanisms with characteristics such as responsiveness, scalability, fine-granularity, reliability, and **Context-Awareness (C-A)** [Oua+17].

Information collected in IoT environments like sensor measurements and actuators controls is mostly used for analysis and presentation to users. Sensing capabilities of end devices enable a broad range of novel service that could be implemented to improve IoT environments' usability and applicability. These improvements lead to the optimization of IoT services by adding intelligence to them through the self-configuration routines researched within A3, increasing the performance, efficiency, and resiliency of IoT systems [Bor14; ČH18]. These sensing capabilities and their aggregation are referred to as IoT system C-A. C-A [Abo+99a] and its application have been researched since the 1990s in various IT systems (e.g., desktop, web applications, and mobile applications). Nowadays, C-A attracts the IoT system developers' attention to deploy an ”intelligent” IoT system, able to adapt its behavior based on the current system state [Bor14; BDR07b; KC03]. Beside improving IoT systems through adding the ”intelligent” features, C-A also allows IoT security mechanisms automation through context analysis and provided context information. Proper candidate for IoT security automation is security policies configuration and the maintenance of trust relationships between entities in IoT systems, minimizing human effort required to configure IoT environment's security properties. For that reason, the application of C-A principles in authorization schemes, identification, authentication, and discovery mechanisms, requires future research and development [ČH18].

1.2 Problem Statement

IoT needs to enable seamless connectivity for Things, users, and computing infrastructure to provide intelligent services that are based on networking, sensing, processing, identifying, and visualization capabilities. This requires the research and development of a standard system architecture, accompanied by various properties, such as scalability, security, privacy, interoperability, or C-A. Due to the IoT systems' unique characteristics, primarily the heterogeneity and limited Things' resources, developed solutions have to be resource-efficient and adaptable to various IoT application domains (cf. Figure 2.1).

Things' computational, storage, networking, and energy constraints limit hosting of IoT services like analytics and measurement visualization. Therefore, IoT services are leveraged to CC resources, which offer sufficient resources for hosting IoT platforms. Still, CC introduces several drawbacks outlined in Section 2.2.1, mainly due to the Cloud Servers remote deployment. Deployment of computing, storage, and networking resources at the IoT networks edge layer is a promising approach for overcoming CC drawbacks. Emerging paradigms, such as EC, MCC, MEC, and FC enable hosting of IoT services close to the Things. The ultimate goal of these paradigms is to improve the IoT services' scalability and responsiveness. However, hosting IoT services using these computing paradigms requires the definition of standardized system architecture and deployment methods. In that respect, secure deployment of computing resources at the IoT network's edge becomes an open issue due to the attack plain enlargement through the new services deployment.

Securing IoT systems represents a complex challenge, primarily due to the Things' resource constraints, as well as IoT networks' heterogeneity and dynamicity. For those reasons, the security solutions should be interoperable enough to support various IoT devices and services, but also responsive and lightweight, so that Things can apply them. Moreover, managing trust through mutual authentication represents a vital challenge. Traditional trust management systems, such as Public-Key Infrastructure (PKI), Web-of-Trust, and Key Distribution Center (KDC) fail to support all the IoT requirements. Still, adaptation and extension of these systems can lead to a promising solution for IoT systems.

AC is a critical security mechanism for ensuring data confidentiality and privacy. Since IoT devices fail to host resource-demanding AC services, they are leveraged to CC servers. CC servers' remote deployment introduces delay for access policies validation and hinders the development of adaptable access policies, based on C-A factors in a local IoT network. Thus, the AC services distribution through edge layer computing paradigms, followed by the extension of AC models, would minimize the delay and allow intelligent, dynamic access policies that can adapt to the IoT system's state. This enables IoT security policies automation, reducing the human-effort required for IoT system configuration.

In conclusion, this dissertation develops trust management and AC solutions for IoT systems. It aims at utilizing FC for hosting distributed security services in the IoT network and reducing the gap between resource-constrained Things and remote CC servers. The developed services improve the identity and key management strategies through distribution on FNs. Moreover, distributed AC provisioning, extended through adaptable context-aware security policies are further objective, enabling automated, flexible access policies management, followed by reduced communication and processing delay.

1.3 Research Questions

This dissertation aims at developing distributed security mechanisms for IoT systems based on the FC paradigm. The main goal is to bridge the security services provisioning between resource-limited Things and computationally-rich remotely deployed CC servers. A further goal is to enable the automation of the security services using context information in the IoT environment. To achieve these goals, the introduced research areas (A1, A2, and A3) inspire the definition of the key research question specified in Sections 1.3.1 to 1.3.3, aiming to enable the following features:

1. Secure deployment of FC-based services;
2. Establishing FC-based Identity and Trust Management for IoT;
3. AC distribution and its extension through C-A.

1.3.1 *Fog Computing-based Access Control Distribution*

Distribution and deployment of AC provisioning close to the Things brings several benefits to the IoT, such as reduced communication and processing delay or increased reliability of AC mechanisms in low network QoS scenarios. However, shifting AC from CC servers to FNs brings multiple challenges like:

- Establishing and maintaining trust in AC gateways;
- Distributed user session management;
- Decentralized security policies management.

AC distribution is a heavily researched topic, with several standard protocols and frameworks (e.g., OAuth2, Shibboleth, and OpenID Connect). Still, defined FC requirements for ”offline” use-cases (FC services availability despite missing internet connection) require further research in this area. In order to decentralize AC provisioning using FC, this dissertation deals with the following research question, accompanied by two research subquestions:

(RQ 1) How can decentralized management of access control be achieved in a Fog computing environment?

> **(RQ 1.1)** Which techniques and technologies can be used in order to keep a consistent state of an access control framework in the hierarchical IoT organization: Cloud Server – Fog Node - Things?
>
> **(RQ 1.2)** What kind of trust connections between authentication and authorization entities need to be achieved to minimize attack surfaces on end devices with regard to access control?

Answering **RQ 1** involves the analysis of different AC aspects addressed by RQ 1.1 and RQ 1.2. Firstly, as emphasized in **RQ 1.1**, distribution of security policies management is analyzed for IoT use-cases (cf. Section 3.3.2), focusing on AC schema modeling and enabling fine-granularity and extensibility for C-A factors. The designed model is presented in Section 3.3.1, along with implementation (cf. Section 4.6.1) details and performance evaluation in Section 5.4.2. Secondly, local AC provisioning through the AC components deployment on FNs requires managing trust relationships between Cloud Server and FNs. This is examined in RQ1.2 for the following areas: (i) FC deployment requirements (cf. Section 3.1.2), (ii) trust modeling (cf. Section 3.2.2), and (iii) distributed AC (cf. Section 3.3.2). The proposed AC distribution model enables mutual authentication between FC-and CC-based AC components, allowing security policies enforcement close to the Things, reducing processing and data transmission latency. Extensive insights on the distribution model are documented in (i) Sections 3.1.3 and 3.2.2, with regard to the trustworthy FC-based security services deployment and (ii) Sections 3.3 and 3.3.2, concerning distributed approach for AC configuration. Implementation details on trust-related bootstrapping and digital certificate management procedures are provided in Sections 4.4.2 and 4.4.1, respectively. The implementation of the distribution mechanisms for the AC configuration is presented in Section 4.6.3. Finally, functional evaluation of the trust management and the AC distribution is provided in Sections 5.3.1 and 5.4.1, respectively.

1.3.2 *Fog Computing-based Identity and Trust Management*

Building and maintaining trust in IoT systems poses several research challenges to be resolved, primarily on how the Things can be uniquely identified and which mechanisms can be used in IoT to build trust relationships based on mutual authentication. The diversity of Things, vendors, and their computational capabilities represents a critical factor in finding a common approach for identifying and authenticating Things. Also, Things' computational limitations hinder the application of traditional encryption algorithms and key distribution protocols (e.g., RSA, TLS, Diffie-Hellman, or Qu-Vanstone). Moreover, IoT networks' scale and dynamics, with nodes often joining and leaving the network, pose further challenges for managing encryption key and overall trust in the IoT networks. Using the computational power of FNs in local IoT networks to shift IdM and encryption keys management from Cloud to Fog significantly simplifies overall trust management in IoT and enables the local trust circles creation. Achieving these results in the following research questions:

(RQ 2) How can Fog computing be used for improving Identity management and authentication in IoT systems?

> **(RQ 2.1)** How can computational power on the edge of the network be used for improving scalability of automatic (e.g., PKI or Web of Trust) authentication procedures?
>
> **(RQ 2.2)** How can Fog nodes be efficiently and effectively used as secure storage of identities for resource-constrained devices and providing mutual authentication in IoT environments?

As described in **RQ 2**, the research examines challenges in (1) IdM and (2) mutual authentication, leading to the trust establishment and maintenance in IoT systems. The envisioned FC-based trust management framework developed for answering **RQ 2**, **RQ 2.1**, and **RQ 2.2** incorporates the trustworthiness of Things, FC-based IoT services, and IoT users. Background on the IdM topic is provided in Section 2.3.3, which is used to design and implement identity naming schema presented in Section 3.2.1. This schema provides structured notation for building identities in IoT environments, also incorporating information about the entities' position in the IoT system's Cloud - Fog - Thing hierarchy. Enabling mutual authentication in IoT required analysis of models for building and managing trust (cf. Section 2.3.2). Thereby, the critical topic is the application of encryption algorithms and Key Management Protocol (KMP), mostly due to the Things' resource-constraints and scale of IoT networks. Thus, a simulation engine for performance evaluation of various aspects of hierarchical trust management models is designed and implemented as documented in Section 4.5, enabling the choice of the best performing trust management model for the given use-case (cf. Section 5.2). Based on the simulation results, a novel FC-based trust management system (cf. **RQ 2.1**) enabling mutual authentication in IoT networks is designed (cf. Section 3.2.2), implemented (cf. Sections 4.4.2 and 4.4.1), and evaluated (cf. Sections 5.3.1 and 5.3). Moreover, the developed trust management system is extended through security profiles to support identity and key management procedures (cf. Section 4.4.3) for the Things that are not capable of supporting digital certificates, enabling E2E identity and trust management mechanisms in IoT (cf. **RQ 2.2**).

1.3.3 *Context-aware Access Control for IoT*

Deployment of FNs close to the Things allows processing the information flowing in the local IoT environment, leading to the local IoT network's state and context analysis. This allows building intelligent IoT services, automatically adaptable to various IoT scenarios. Automation of configuration mechanisms for security policies can benefit from context analysis in IoT, reducing the required human effort to set up a security configuration. Moreover, C-A enables a more informed decision-making process during security policy execution. Since AC mechanisms are the central point for configuring and applying security policies, their integration with content analysis would enable adaptable, "smartified" security policy management. To achieve this, various C-A sources have to be examined concerning the context information they are analyzing and integration points with established AC models. This results in the following research questions that are investigated in this dissertation:

(RQ 3) How can context-awareness be incorporated in access control of a Fog computing based IoT system?

(RQ 3.1) Which factors can influence access rights validation and adjustments in a context-aware IoT environments?

(RQ 3.2) How can these factors be embedded in today's access control models?

Providing answers to **RQ 3** involves analyzing context information management approaches (cf. RQ 3.1) and steps to integrate context information into AC services (cf. RQ 3.2). Firstly, RQ 3.1 analyzes how context can be quantified and managed in computer systems. This involves the identification of context sources and context information collection, processing, and distribution procedures (cf. Sections 2.5.1 and 2.5.2). Secondly, strategies for building C-A systems have been evaluated in Sections 2.5.3 and 2.5.4, leading to the design requirements and decisions for developing the C-A AC solution. Primary contributions of the developed C-A AC solution involve (i) defining Application Programming Interface (API) and message exchanges for integration of context information into security policies (cf. Section 3.4.1), (ii) defining common C-A AC data model presented in Section 3.4.2, and (iii) extending AC services with capabilities to apply context information during authorization procedures. Finally, the designed solution has been implemented (cf. Section 4.7) and evaluated in the Smart Home environment (cf. Section 5.5), providing results on functional and performance implications of C-A AC services in the IoT environment.

1.4 Structure of this Dissertation

The aforementioned problems are investigated in the defined research questions in Section 1.3 and will lead to an IoT framework –COSYLab–, which is designed, implemented, and evaluated in this dissertation. Thus, the dissertation is structured as follows:

Chapter 2 presents the background information related to the topics relevant to answering the research questions defined in Section 1.3. In the introductory part of Chapter 2, insights on IoT are documented, focusing on IoT Application Domains, Things computational characteristics, and IoT reference model (cf. Section 2.1). Afterward, the role of CC and computing decentralization possibilities through MCC, MEC, EC, FC are being discussed in Section 2.2. Secondly, various trust and Trustworthy Networking (TN) aspects are being analyzed in Section 2.3, with the goal of ensuring CIA securing properties in computer networks. An AC overview is provided in Section 2.4 through its (i) general characteristics, (ii) the main building blocks: Authentication, Authorization, and Accounting (AAA), and (iii) AC distribution approaches. Insights on C-A are presented in Section 2.5, including details on context information identification, collection, processing and distribution strategies, as well as insights on building and integrating C-A in computer systems.

Chapter 3 provides details on solution design requirements and decisions, with the goal of building FC-based security services for trust management and C-A AC. First, Section 3.1 presents general FC architecture requirements and provides decisions on the FC-based security services deployment model, accompanied by a brief description of the deployed services and their interdependencies. Afterwards, guidelines for building trustworthy FC-based IoT network in this dissertation's scope are described in Section 3.2, including design decisions on different aspects of TN: IdM, trust management, and E2E security. Based on the designed trust management model, the AC services deployment model is presented in Section 3.3, also taking into consideration the previously mentioned FC requirements, along with the AC IoT-specific requirements concerning: (i) security policies management, (ii) security policies modeling, and (iii) approach for secure AC distribution in IoT. Finally, guidelines on C-A integration in AC are presented in Section 3.4. These guidelines include insights in: (i) collecting context information using C-A agents, (ii) context information management and modeling, and (iii) its application in AC mechanisms - security policies management and authorization.

Chapter 4 presents the implementation details addressing the problem statements mentioned in order to answer the defined research questions RQ 1 - RQ 3 in this dissertation. Section 4.1 provides an overview of the COSYLab IoT framework. Section 4.2 presents the overall solution's software architecture describing used technologies, developed software components, and the integration interfaces. Details on FC services deployment are provided in Section 4.3. TN services are described concerning various trust management aspects: identity and certificate management, trust bootstrapping procedures, and establishing E2E security (cf. Section 4.4). Moreover, trust models' efficiency is examined using the developed trust management performance simulator, for which the details are provided in Section 4.5. AC services implementation (cf. Section 4.6) is described through details on the developed components and considerations of the applied AC model, followed by the procedures for AC configuration based on synchronization between the CC server and FNs. Finally, AC services extension through C-A is described in Section 4.7, providing details on the messages exchanged between C-A agents and AC components. The messages describe how the context information is managed and applied in the AC service in the scope of the proposed framework.

Evaluation of the developed solution is presented in **Chapter 5**. The solution is verified concerning its performance, functionality, and fault-tolerance, involving four different solution areas: FC, TN, AC, and C-A. The performance and resource consumption overview of the deployed security services on FN is provided in Section 5.1, verifying the feasibility of the FC-based service deployment in the Smart Home framework. Afterwards, simulation-based evaluation of different PKI schemes is described in Section 5.2, leading to the results for choosing the best performing PKI scheme in the given IoT application domain - Smart Home. Based on these results, the implemented TN services are evaluated in Section 5.3, focusing on the services' performance and scalability concerning identity, certificates, and trust management. The AC services' evaluation (cf. Section 5.4) presents

(i) a proof of operability for the security policy configuration synchronization and (ii) the authorization performance concerning processing latency. A C-A integration evaluation is presented in Section 5.5, following similar approaches used for AC evaluation. For that purpose, a proof of operability is provided for access policy validation using context information. Additionally, the computation and performance overhead for context-based access policies has been evaluated. Finally, the developed solutions' fault-tolerance and error-handling procedures have been analyzed and presented in Section 5.6.

Finally, **Chapter 6** provides a summary of the outcomes for the defined research questions in this dissertation. Firstly, the achieved contributions in areas TN, AC, and C-A are summarized, and answers to the defined research questions concerning using FC for deployment security services in IoT are given. Lastly, an outlook for future developments in the involved research areas is provided.

2

State of the Art & Related work

The research goals defined in Section 1.3 involve research in multiple areas. For that purpose, this chapter provides state-of-the-art in these areas. Firstly, IoT basics are presented in Section 2.1, followed by the IoT deployment and distribution approaches in Section 2.2. Afterwards, relevant areas of network security are analyzed: building trustworthy networks in Section 2.3 and enabling AC in distributed systems in Section 2.4. Section 2.5 provides an overview of strategies for embedding context information in IoT systems, leading to the creation of intelligent IoT environments. Lastly, Section 2.6 summarizes the chapter and outlines the research gaps relevant to achieving this dissertation's goals.

2.1 IoT basics

The IoT is a novel paradigm that enables a set of new services and technical innovations in IT. Amongst numerous definitions, IoT is defined as "*a global infrastructure for the information society, enabling advanced services by interconnecting physical and virtual things based on existing and evolving interoperable information and communication technologies*" [22p]. The goal of IoT is the seamless integration of physical devices (e.g., power outlets, thermometers or smart watches) into the connected world and enriching them with sensing, actuating, and communication capabilities. Through the interconnection of devices, IoT builds global infrastructure for the information society, allowing the introduction of novel, advanced services for end-users. Therefore, the essential building block for IoT is a Thing, representing an entity of the physical world –physical device– or information world –virtual Thing– with the capability of being integrated and identified in communication networks. [22p]

In order to incorporate physical devices into IoT environments, they are extended by additional computing and networking equipment like CPU, memory, and antennas. Through the extension, physical devices are mapped to virtual Things and are provided with support for the required IoT capability - communication, as well as the optional ones: sensing, actuation, data capture, data storage, and data processing. Due to the scale of IoT systems, Things' size and weight, available power, and energy, as well as economic reasons, Things are mostly equipped with low-constrained computational resources, in literature also called *Constrained Node (CN)* or *Constrained Devices* [BKC20]. These constraints lead to the severe operational boundaries of Things: processing cycles and computing power, programming code space, available power, user interfaces, firmware upgradeability, and networking connectivity concerning protocols on different ISO/OSI layers. CNs are categorized within several categories, as presented in Table 2.1. [BKC20]

Table 2.1: Classes of CNs[8][BKC20]

Name	data size (RAM)	code size (Flash)
Class 0	« 10 KiB	« 100 KiB
Class 1	10 KiB	100 KiB
Class 2	50 KiB	250 KiB
Class 3	100 KiB	500 - 1000 KiB
Class 4	300 - 1000 KiB	1000 - 2000 KiB
Class 10	4 - 8 MiB	» 2000 KiB
Class 15	0.5 - 1 GiB	» 2000 KiB
Class 16	1 -4 GiB	» 2000 KiB
Class 17	4 - 32 GiB	» 2000 KiB
Class 19	» 32 GiB	» 2000 KiB

8 KiB = Kilobyte, MiB = Megabyte, GiB = Gigabyte

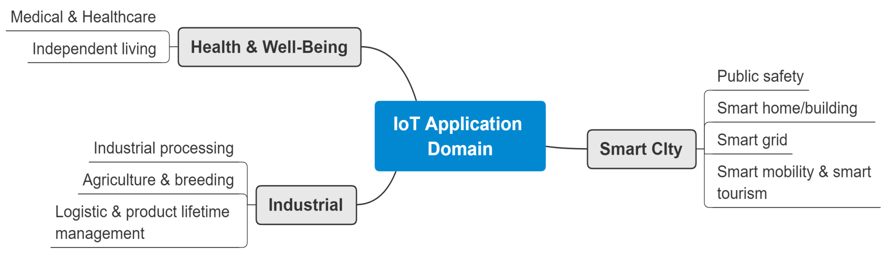


Figure 2.1: IoT application domains [Bor14]

The diversity of Things allows numerous types of innovative services that can be offered to the end-users, leading to the everyday life enhancement, as well as to a positive impact on socio-economic factors. These services can be divided into three major IoT application domains (cf. Figure 2.1): (i) Industrial domain, (ii) Smart City domain, and (iii) Health & Well-being domain. Even though the domains are divided into separate categories, their functionalities overlap to some degree, since they serve the common purpose of improving end-users' life quality. Each major IoT application domain contains multiple application areas, and further offers a limitless and increasing number of IoT services in its particular IoT application domain. [Bor14]

The diversity and complexity of enabling technologies and IoT application domains are identified as one of the vital challenges for the future development of IoT systems. Hence, the standardization of IoT systems' architecture and technologies is seen as the driving factor for future IoT applications. These standards should offer a common ground and architecture for building interoperable, seamlessly connected IoT systems, enabling simplified integration of different IoT application domains, leading to a higher number of IoT services. [ČH18]

An open IoT architecture should offer a framework for the integration of various technologies so that they are fully interoperable between IoT application domains. Accomplishing that requires an open IoT architecture that supports heterogeneity of Things, underlying computer networks, data, and applications. Nonetheless, to support analytics, system's intelligence, and user-friendly IoT applications, this open IoT architecture should enable interactions across IoT application domains, as well as scalable IoT services management [ČH18]. With these expectations in mind, standardization bodies (e.g., IETF or IEEE) introduced several conceptual IoT architecture models, mostly following the approach using layered frameworks and architectures as depicted in Figure 2.2 [Sar+15].

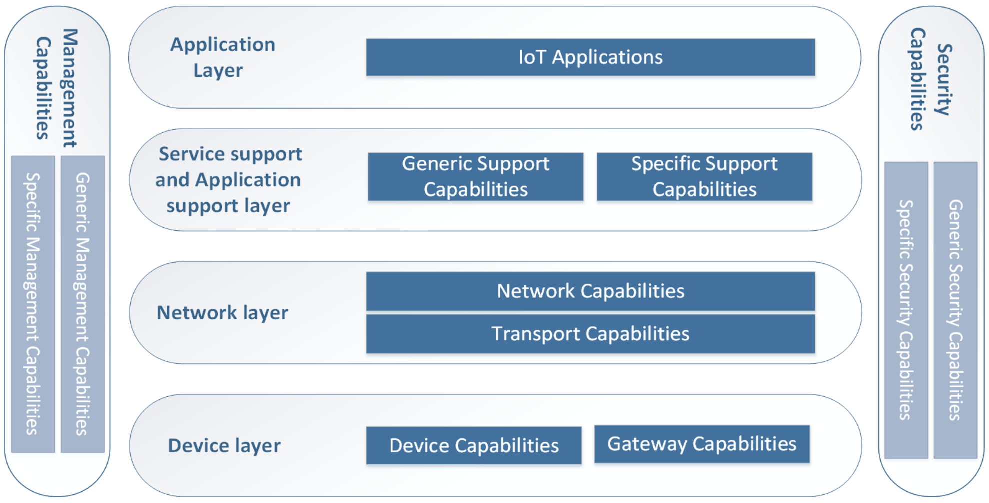

Management Capabilities
Specific Management Capabilities
Generic Management Capabilities
Application Layer
IoT Applications
Service support and Application support layer
Generic Support Capabilities
Specific Support Capabilities
Network layer
Network Capabilities
Transport Capabilities
Device layer
Device Capabilities
Gateway Capabilities
Security Capabilities
Specific Security Capabilities
Generic Security Capabilities

Figure 2.2: IoT reference model [22p]

Each layer has a special purpose:

- *The Application layer* represents a placeholder for IoT applications;
- *The Service support and application support layer* provides the required support for applications, grouped in two capabilities sets:
  - The Generic capabilities involving common capabilities which can be shared between IoT applications, i.e. data processing and storage capabilities;
  - The Specific capabilities, dedicated and tailored for the particular IoT application requirements.
- *The Network layer* provides network connectivity and access and transport control functions, as well as the IoT service data transport;
- *The Device layer* lists capabilities for IoT devices to support, divided into two categories:
  - The Device capabilities include the support for interaction with the communication network (direct and indirect), ad-hoc networking and energy-saving sleeping and waking-up mechanisms;
  - The Gateway capabilities incorporate support for multiple network interfaces (e.g., ZigBee, Bluetooth, and Wi-Fi) and network protocols conversion.

Management Capabilities of the IoT reference model cover fault, configuration, accounting performance, and security capability classes. These classes can be categorized into generic and application-specific ones. Examples of management capabilities are device management (remote device activation/deactivation and firmware upgrade), network management, and traffic management (congestion avoidance and resource reservation).

Security capabilities aim at providing security mechanisms on different IoT reference model layers:

1. AC, data confidentiality, privacy, malware protection on the Application layer;
2. Traffic signaling data confidentiality and integrity, network traffic AC on the Network layer;
3. Device integrity validation (software and hardware), device AC on the Device layer.

2.2 Cloud & Fog Computing in IoT

IoT enables a sharply increasing number of innovative services. However, Things may not be capable of hosting the implementation of those services due to their available computational capabilities. For that reason, IoT service provisioning has been leveraged to the CC-based systems, overcoming the issues with battery-powered, low-computational resourced Things [Pul+19]. The implications of CC on IoT systems and accompanying benefits and drawbacks are presented in the following sections. Moreover, to overcome negative aspects of the CC, innovative computing paradigms based on CC decentralization are enlisted and described.

2.2.1 *Cloud Computing Basics*

In the previous years, CC enabled sufficient flexibility for IoT systems by offering multiple Cloud service deployment models like IaaS (Infrastructure-as-a-Service) [Sha+17], PaaS (Platform-as-a-Service) [BMW11] and SaaS (Software-as-a-Service) [Cus10]. These models introduced different possibilities for the IoT vendors to deploy their services and make them accessible for the end-users, especially to monitor and manage systems and actors in Smart Home applications.

CC platforms are mostly deployed in a DC, incorporating numerous processing units interconnected in high-speed, high-bandwidth computer networks [Sha+17]. DCs function as a central-point for data acquisition, processing, and storage. DCs can be public, private, and hybrid. The most distinctive characteristic of public DCs is that services and resources are provided by a third-party to anyone who requires them. In contrast, private DCs [Sot+09] are driven by specific companies,

Table 2.2: IoT application domain latency requirements [Sch+17]

Domains	Latency Request
Industrial processing	0.25 - 10 ms
Smart grid	3 - 20 ms
Smart mobility	10 - 100 ms

and their services are reachable only to the employees of that company. Hybrid DCs represent a mixture of public and private DCs, where some business processes are deployed in public DCs. In contrast, privacy and security-critical processes are deployed within private DCs.

This Cloud service idea is a driving factor for the development of the existing IoT solutions. Despite flexibility for the deployment of Cloud services, DCs still represent a centralized entity for the data processing introducing at the same time a Single-Point-of-Failure (SPoF) to the total system. Further drawbacks are known and are depicted at the end of Section 2.2.1 in detail.

The physical distance between CC services and IoT devices, as well as the centralized nature of DCs, introduces **unpredictable network and data processing latency** in IoT [Sat15; RLM18]. Multiple IoT application domains, like the ones listed in Table 2.2, require Ultra-Reliable Low-Latency Communications, meaning that the data communication links between processing unit for IoT services and Things must ensure low and predictable response times. High communication latency between Things and DCs becomes challenging for meeting the identified latency requirements. For example, the Amazon Web Services network latency map [22h] has been analyzed for the DC in Virginia (USA). The gained results show that in most cases the average network round trip time is higher than 100 ms (e.g., 258 ms to the DC in Beijing, 125 ms to Italy or 146 ms to Brazil). Average round trip times of below 100 ms, which could satisfy IoT latency requirements, are present only in North America and some parts of Western Europe.

As multiple IoT application domains (e.g., Smart Home and eHealth) produce a significant amount of users' private data [Por+16], securing that information plays a vital role in the further acceptance of IoT systems. An obvious choice for ensuring data protection is to encrypt the data before transmission over the Internet. However, CNs at the edge of the IoT network are often incapable of performing computationally demanding operations required for symmetrical and asymmetrical encryption. Thus, users' private data transmission to the DC can be eavesdropped and altered, therefore endangering TN concepts (cf. Section 2.3), as well as **data privacy and security**, allowing network attacks, such as Man-In-The-Middle, Data Leakage, and Replay Attack [Zho+17]. Moreover, IoT end-users are mostly unaware of the exact location where their private data is being stored due to the global coverage of Cloud services. Therefore, it cannot be guaranteed that the data privacy regulations applied in the DCs are equal to the ones in the user's country. For example, the difference between General Data Protection Regulation (GDPR)[9] in Europe and the California Consumer Privacy Act (CCPA)[10] in the USA could lead to the users' lack of awareness for their data privacy settings.

The exponentially increasing number of deployed Things sets additional burden on computer networks, demanding big amount of bandwidth to transmit produced data to data processing units - DCs. International Data Corporation[11] estimated that the data generated from IoT devices will reach 73.1 zettabytes (1 zettabyte = 1.073.741.824 terabytes) by 2025. Such an envisioned development puts a great demand on provisioning sufficient network bandwidth to support future IoT systems while ensuring other application domains for IP networks (e.g., Video Streaming and Voice-over-IP) to retain existing QoS. At the same time, novel intelligent IoT services will require additional processing of the produced data, leading to the **increased bandwidth consumption** towards DCs, as well as the **higher dependence on network QoS**. This can represent a two-edged sword since IoT systems are also to be deployed in hostile environments or rural areas with weak networking infrastructure, meaning that network QoS will be hard to guarantee, leading to interrupts in IoT service provisioning, due to the lack of DC reachability. [Sat+13]

Deployment of Things in various environments enables better environments' observation and monitoring, therefore providing a possibility for making more-informed decisions and optimizing the overall system's safety and performance. Environment's monitoring is defined through the terms *Context* and *C-A*

9 https://op.europa.eu/en/publication-detail/-/publication/3e485e15-11bd-11e6-ba9a-01aa75ed71a1/, last access May 2, 2022
10 https://oag.ca.gov/privacy/ccpa, last access May 2, 2022
11 https://www.ridge.co/blog/iot-and-the-cloud/, last access May 2, 2022

[Abo+99b]. In terms of IoT environments, local context represents locally sensed and measured information (e.g., traffic statistics, network conditions, and locally discovered Things and services) that is locally aggregated and processed to deduct the current system's state. However, primarily due to the geographical distribution and lack of proximity between local context and DC, limited context information is shared between them, which limits the capabilities of Cloud Services **to directly obtain and process all local contextual information** [Pul+19; RLM18].

Due to the aforementioned drawbacks and limitations, CC imposes several limitations for the novel IoT services development and adoption. Therefore, significant research effort has been invested in the previous years with the goal of overcoming limitations introduced by CC [Sat+13; Pul+19; RLM18; Bon+12b]. The mainstream of these efforts was aimed towards the distribution and decentralization of computing and storage resources. This involves placing Cloud services and resources closer to the end-devices and the end-users, i.e. on the or close to the edge of the network, with the ultimate goal of reducing the physical distance due to the centralized nature of DCs.

Moreover, it is worth noting that the drawbacks of CC do not necessarily affect only IoT systems, but rather all services that rely on CC, such as World Wide Web (WWW), MCC Services, Video Streaming Services, Voice-over-IP. Therefore, numerous research contributions with regard to the decentralization of CC services in those areas have been achieved and applied. Content Delivery Networks were introduced as a solution for distributing content to the Content Delivery Network edge servers, improving the overall system's scalability [RXA04]. Furthermore, combining CC and Peer-to-Peer protocols led to establishing decentralized Cloud architectures [Gar+15]. These architectures diverge from the standard Cloud architectures (e.g., Client-Server and Server-to-Server), bringing benefits to both Peer-to-Peer and CC based systems. On the one hand, CC provides stable resources for Peer-to-Peer networks when needed, while on the other hand, Peer-to-Peer networks lower operating costs and load on DCs [TDM10].

In the IoT domain, the distribution and decentralization of CC services on the edge of the network leads to the establishment of several novel computing models. These models share a common idea - deploy CC-like capabilities on the devices - Edge Data Center (EDC) as closely as possible to the edge of the network. Introduction of EDC changed the standard 2-Tier IoT deployment model: ”Cloud - Thing” by introducing the middle layer and therefore establishing the 3-Tier

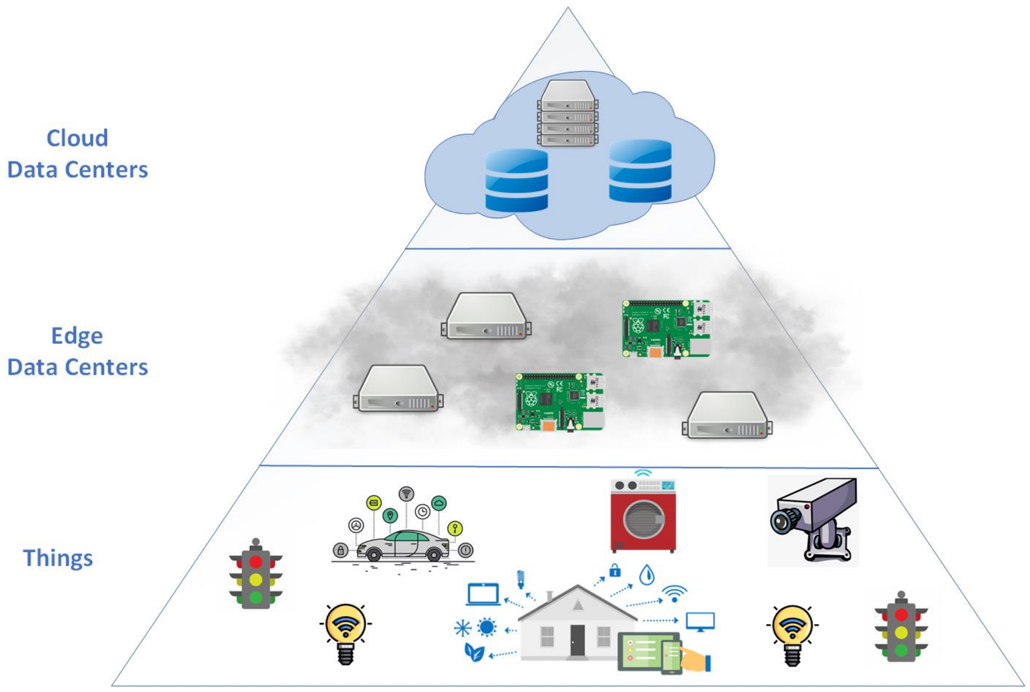


Figure 2.3: Cloud - Edge - Thing deployment model

deployment model: "Cloud - Edge - Thing" (cf. Figure 2.3). Still, several factors differ in these models, especially concerning mobility of EDC, integration with DC, and ownership of EDC [RLM18]. The most cited edge deployment models will be briefly described in the remainder of this section.

The initial deployment model introduced for computing capabilities distribution is MCC [FLR13], which is primarily developed for offloading intensive computing and storage capabilities from battery-powered mobile devices (e.g., mobile phones and tablets) to DCs, leading to a longer battery life in these devices. In the original MCC concept, only centralized CC services were considered as the most promising solution for offloading computational tasks [Ali09]. However, this concept suffered from limitations described above, especially from unpredictable and high network latencies. This motivated researchers to expand the scope of CC services, and leverage computational tasks to Cloudlets, representing EDC in the MCC deployment model [Bah+12]. A Cloudlet is defined as *"a trusted, resource-rich computer or cluster of computers that's well-connected to the Internet and available for use by nearby mobile devices"* [Sat+09]. This expansion provided an underlying infrastructure for the introduction of novel services, such as mobile learning or mobile healthcare [Rah+14], and gave further momentum for the decentralization of CC-like capabilities.

MEC, based on MCC, was primarily adopted and developed by cellular network providers. Namely, in MEC, Cloudlets have been firstly deployed by IBM and Nokia Siemens Network within a mobile base stations [Bec+14]. Afterwards, in 2014, the ETSI created an Industry Specification Group (ISG) with the final goal of defining and integrating a standard MEC into cellular networks, called ETSI MEC [22e]. Afterwards, ISG group researched applicability of MEC on the non-cellular IT systems, resulting in an update of the previous standard and renaming MEC to Multi-Access Edge Computing [22f]. Even though MEC is technologically very similar to MCC, the major distinction between these two deployment models is the ownership of Cloudlet - MCC is envisioned as the model where private entities and individuals pose Cloudlets, whereas Telecommunication companies [RLM18] possess Cloudlets in MEC.

Generalization of the MEC and MCC concept leads to the introduction of EC [Pul+19]. The main idea behind this introduction is widening the applicability of Cloudlets so that they are not necessarily used for mobile applications and cellular network, but also for the other CC application domains like the IoT [Nah+18]. However, conceptually EC does not differ significantly from MCC and MEC since the decentralization of CC-like capabilities remains mostly based on Cloudlets as locally deployed EDC with DC as a fallback option. This characteristic of the EC still represents Cloudlets as SPoF concerning high performance and low-latency requirements.

FC was introduced as a novel deployment model [Bon+12b] improving decomposition and distribution of CC-like capabilities on a hierarchy of nodes, also including the DC. FC is considered an CC extension, with the goal of providing computation, networking, and storage resource between Things and traditional Cloud Servers. Therefore, compared to EC, FC is rather conceptualized as a complementary extension of CC, offering CC service closer to the edge of the network, across multiple, hierarchically-organized devices. Furthermore, the lack of hard-requirement for provisioning all CC services on each FC EDC allows additional flexibility for deployment of FC-based networks and simpler decomposition of CC services [Pul+19]. Finally, FC is focused on optimization of the network infrastructure and increasing utilization of existing devices in the network, while EC is focused on providing CC-like services deployed on the devices dedicated for that purpose, namely Cloudlets.

2.2.2 *Fog Computing Basics*

Initially conceived as a distributed computing paradigm where processing is done at the edge of the network, FC represents a promising opportunity for enhancing the scalability of the current CC-based systems. Bonomi et al. provided the first definition of FC [Bon+12b]: *"Fog computing is a highly virtualized platform that provides compute, storage and networking services between end-devices and traditional DCs, typically, but not exclusively located at the edge of the network"*. In this definition, the authors emphasized several points that are vital for the overall application of FC:

- **Highly virtualized platform** offers flexibility for deployment of CC services on any devices with sufficient storage, computing and networking capabilities;
- **Between Things and DCs**, stating that FC does not neglect the role of CC, but embodies FC as middle part in Thing - FN - DC continuum;
- **Typically, but not exclusively at the edge of the network** imposing non-flat, hierarchical deployment model for the device offering FC services.

These three points state that FC as a paradigm is fully complementary to the CC and can be seen as its extension. Moreover, compared to the EC paradigms, flexible hierarchical FC deployment models focus more on the infrastructure side, while EC focuses more on the Things' side [Shi+16]. However, the paper providing FC definition [Bon+12b] lacked the definition of Fog Devices or FNs, which was later identified as important for the description of FC-based systems. FN is defined as *"a physical device where Fog Computing is deployed"* [YLL15]. Merging this definition with the initial definition of FC, FN can be defined as **any physical device with sufficient storage, compute, and networking capabilities for provisioning highly virtualized deployment of the FC-based services**.

Being initially defined in 2012, FC is still considered as being in its infancy phase. Therefore, while being applied in several IoT application domains (cf. Section 2.1), standardization efforts for FC architecture are in progress, and the first standards for FC architecture are published. The main effort within the FC standardization has been provided by the OpenFog Consortium, which has been founded in 2015 by ARM, Cisco, Dell, Intel, Microsoft, and the Princeton University and later integrated with the Industrial Internet Consortium[12]. Their main objectives are:

12 `https://www.iiconsortium.org/press-room/12-18-18.htm`, last access May 2, 2022

- Creation of open, comprehensive architecture for the FC;
- Promotion of the FC adoption in the several application domains that may benefit from it;
- Accelerating the development of FC standard along with standardization bodies.

In 2017, the consortium published the OpenFog Reference Architecture (OFRA) [22g], describing the initial FC architecture and framework for the development of interoperable platforms using FC. In 2018, OFRA was accepted as standard by IEEE SA[13], and published as IEEE 1934 [22a].

In [22a], the authors describe multiple high-level aspects of the FC architecture: *Pillars*, *Perspectives* and *Views*. *Pillars* describe the set of driving, core principles for OFRA. They represent *"key attributes that a system needs to embody the OpenFog definition of a horizontal, system-level architecture that provides the distribution of computing, storage, control, and networking function closer to the data source (users, Things, et al.) along the Cloud-to-Thing- continuum."*. The *Pillars* have been divided into several categories, each describing particular set key of requirements that OFRA-based FC systems need to satisfy: (1) Security, (2) Scalability, (3) Openness, (4) Autonomy, (5) Programmability, (6) Reliability, Availability, and Serviceability, (7) Agility, and (8) Hierarchy.

Beside the *Pillars*, OFRA defines *Perspectives* and *Views*, providing more concrete description on architectural points for FC-based systems' architects and developers. "*A Perspective is a cross-cutting concern of the architecture*", meaning that it interweaves multiple layers of the OFRA. In OFRA, layers are described through *Views*, which are defined through"*a representation of one or more structural aspects of the architecture*". Currently, in OFRA, there are three specified *Views* described:

1. The **Software View** defines required characteristics for services and applications running on a FN;
2. The **System View** defines points that system architectures need to address while designing FC-based systems;
3. The **Node View** defines characteristics that a chip installed in a FN should possess.

Additionally, *Perspectives* defines technical and non-technical capabilities of the FC systems, which ought to be supported by *Views*:

13 https://standards.ieee.org, last access May 2, 2022

1. The **Performance and Scale Perspective** targets at improving overall system performance (e.g., reducing network and computational latency);
2. The **Security Perspective** aims at providing E2E security in Cloud-to-Things continuum;
3. The **Manageability Perspective** defines requirements for dynamic management of all aspects of Fog deployments;
4. The **Data, Analytics and Control Perspective** describes localized data capturing, storing, analysis, and control;
5. The **IT Business and Cross Fog Applications Perspective** describes interoperability of FC-based services at any level of Fog hierarchy in multi-vendor systems.

The aforementioned characteristics of CC decentralization deployment models, especially FC, represent the driving factor for the establishment of a new breed of IoT services. Envisioned as an extension of CC, IoT systems can benefit from FC as an intermediation layer between DC and Things. Furthermore, highly-virtualized deployment of FC services enables the hierarchical organization of IoT services, leading to the better utilization of computational resources in local IoT networks, as well as providing CC-like services with the small distance from Things. The hierarchical organization of FNs depends on multiple factors [Bel+19]:

1. Complexity and computational requirements of the offered service;
2. Computational capacity of the FNs;
3. Functional requirements of the IoT application domain.

In order to achieve correct system's operation and overall end-user satisfaction, IoT systems pose various key requirements to the accompanied environments providing computing capabilities for Things. These requirements are published in [Bel+19], taking into consideration multiple surveys and standardization proposals for FC and refining them by considering multiple IoT application domains. A summary of these requirements, followed by a brief definition for each one, is given in Table 2.3.

The distributed nature of FC proves to be a promising approach for solving low-latency, bandwidth-independent, location-aware requirements for the introduction of novel IoT services and applications. The following paragraphs of this section list benefits of the FC application onto multiple IoT application domains.

Connected vehicles and Smart Transportation Systems can employ IoT and FC services for broader information exchange and data-analysis enrichment to resolve several transportation issues in highly-urbanized environments, such as traffic congestion, time losses, accidents, and pollution. For this purpose, low-

Table 2.3: IoT key requirements for FC [Bel+19]

Requirement	Definition
Scalability	Capability of the system to scale in relation to the quantity of managed information
Interoperability	A standardized way to describe and exchange information, together with an abstraction layer that hides physical differences among elements
Real-time responsiveness	Low-latency communication and real-time interactions between elements
Data quality	Differentiating rich-information data, filtering out faulty or noisy data, as well as aggregation and integrated use of data coming from highly heterogeneous and different IoT sensors/actuators.
Security	Provisioning CIA for IoT environments.
Location-awareness	Identification of the location of the deployed applications and using of this information to improve the data processing and adaptation modules.
Mobility	Supporting mobile devices to shift from an FN authority to another without interrupting systems operations or causing any problems.
Reliability	Multiple categories: Resiliency against the failure of any individual FN, failure of the whole network, failure of the service platform, failure of the user's interface connected to the system.

latency data processing and analysis, as well as location-awareness, are required, being the stumbling blocks for the employment of CC. However, FC can have a crucial role in such scenarios [KCT16]. Deployment of FC can occur through the installation of FNs on static (e.g., road infrastructure or traffic lights) and mobile entities (e.g., cars or trucks), enabling connectivity and interaction scenarios: cars to cars, cars to access points (static entities) and access points to access points [Bon+12b]. Through that deployment, smart transportation systems can benefit from [22g]:

1. Decreased bandwidth consumption;
2. Data processing units deployment in the intermittent presence of vehicles;
3. Provisioning location and C-A services to the vehicle in their proximity (e.g., alerting traffic congestion or reporting accidents to the vehicles in the close area).

Smart Grids emerged through the integration of traditional electrical grids and information systems, conceiving services for more efficient, reliable, and secure energy management [Fan+12]. Being the highly distributed system of power supply, sensing, and actuating units, Smart Grids commonly cover the area of hundreds of square miles, resulting in the production of a vast amount of data that saturate the network, storage, and processing units [Pul+19]. Furthermore, Smart Grids require low and predictable response times - up to 20ms [Sch+17], as well as enhanced privacy settings, since the smart metering devices can be exploited to leak personal information about the inhabitants in an area [OO16]. For those reasons, the deployment of FC in Smart Grids can bring multiple benefits, such as (1) achieving low and predictable service response times and (2) performing data aggregation, analysis, and anonymization in the FNs deployed along Smart Meters. [Pul+19]

Smart Healthcare represents the most delicate IoT application domain, since it can profoundly affects peoples' Quality of Living and their lives themselves. Moreover, since Smart Healthcare handles the most sensitive information about the patients, special attention has to be given to the privacy settings and information security in such systems. For those reasons, FC can be a promising approach for many Smart Healthcare issues [Far+18]. Some of the beneficial aspects of FC in Smart Healthcare [Pul+19] are:

1. Low response time, representing the crucial factor in life-critical situations;
2. Robust services provisioning, independent on network QoS;
3. Local protection of the sensitive patient data in FNs (e.g., in a hospital or retirement home) instead of sending them over the Internet to the DC.

Smart Home, Smart Building, and Smart City incorporate management and monitoring of various Things in indoor and outdoor scenarios whilst providing adaptive and intelligent services for improving the system's performance, reducing maintenance costs, and increasing overall Quality of Life for the people residing therein. Since home and corporate IoT deployments involve multiple diverse appliances (e.g., home gateways, routers, and set-top boxes), making use of their computing capabilities within the virtualized FC-based environments would bring several benefits for future IoT systems [Val+16]. Provisioning of IoT services on FNs in local Smart Home and Smart Building context would dramatically increase resilience when Internet connectivity is absent, decrease network latency and bandwidth consumption. Lastly, by keeping sensitive information of the inhabitants in these IoT environments in a local context that is on the FN, the overall IoT system's privacy would be highly improved [Pul+19].

2.3 Trustworthy Networking

Conceptually, trust represents *"measurable belief and/or confidence which represents accumulated value from history and the expecting value for the future"* [22n]. Being influenced by multiple factors (e.g., reputation of entities, relationships between them, decision making, or human decision) quantification of trust is a complex process. For that reason, once building a trustworthy IT system, establishing trust does not involve merely devices and network, but also human trust in the IT system. In the scope of this thesis, building and establishing trust in computer networks is in focus, that is, building a Trustworthy Networking Domain (TND) [22o]. Trustworthy Networking (TN) incorporates a set of methods with the goal of provisioning reliable and secure communications between network entities (trustor and trustee) that have established trust relationships [22o]. These entities are grouped within a TND, characterized by administrative features with a certain trust level and established mutual trust relationships between all of its members.

In the context of TN, establishing secure communication requires the support of several fundamental features [22o]:

1. **Identification** is the process of recognizing an entity in a particular domain as distinct from other entities [22c];
2. **Trust evaluation** represents a decision-making process based on an expectation and belief for a trustee to perform a particular action;
3. **Trustworthy communication** enables the establishment of a secure and reliable link between trustor and trustee.

Application of these features between multiple network entities encapsulates them in the TND, with the common trust policy and security configuration specific for that TND. Within the TND, the following TN features are provided: (i) support for trust management, (ii) continuous network entities trust evaluation, (iii) interface for communication outside of the TND, (iv) unique entity identification using identifiers, and (v) establishment of trustworthy communication links between entities. This encapsulation enables the minimization of attack vectors and simplifies the establishment of mutual trust between entities due to the smaller number of entities in a single TND compared to public networks. Moreover, access to the entities is restricted through a TND interface, which enforces predefined procedures for identification and trust evaluation for the external TND entity. Before joining the TND, an entity is forced to pass a particular trust procedure to establish (also called bootstrap) initial mutual trust with other TND members. Once passing the initial validation, an entity becomes a TND member and is continuously monitored for any misbehavior, which could endanger the TND's trust policy. If such behavior is detected, an entity's trust is revoked, and the entity is removed from the TND. [22o]

2.3.1 *Building Trustworthy Networks*

The operation of trustworthy networks provides secure and reliable communication links between uniquely identified, mutually authenticated network peers. Network security defines secured peer communication through the CIA principles. [Sta16] and [22l] provide the following descriptions for the CIA principles:

1. **Confidentiality** assures restricted availability of the private or confidential information to the unauthorized peers;
2. **Integrity** assures that information and programs are altered exclusively in a specified manner by an authorized peer;
3. **Availability** assures that the system works promptly and that authorized users do not suffer from denial of services.

Accomplishing CIA principles leads to a successful defense against the network attacks and to maintaining trust [Shi07]. Accomplishment is implemented through the security services application defined in [22l]. Network security employs encryption to ensure the trustworthiness of the communication links and the transferred data. Encryption algorithms are divided into two main categories - symmetric and asymmetric encryption. Each algorithm is based on the dissemination of security material, i.e. shared secrets, within the TND through the trust management infrastructure. Security material is mostly in the form of encryption keys, credentials, passwords, tokens, or certificates which are used for the creation of encrypted channels for secure information transmission. [Hu16]

Successful provisioning of CIA principles via encryption procedures depends on multiple factors: encryption key and algorithm strength and system-wide secure security material dissemination. The encryption key and algorithm strength disables the attacker to decrypt the ciphertext or discover the encryption key even in the scenario where the attacker possesses multiple ciphertexts of the communication using the same encryption keys. Still, encryption algorithms are publicly available standards, and their secrecy does not guarantee network communication trustworthiness. In contrast, the encryption keys have to be securely shared and managed between and stored on the communicating entities and in the TND. [Sta16; Bar22]

Security material dissemination is achieved using different methods categorized as KMP. Following the TND's and accompanying security material's lifecycle, key management is divided into four phases [Bar22]:

1. The **Pre-operational phase** ensures that the initial secrets and system attributes are being shared in the TND. Result of this phase is the establishment of initial trust relationships between TND entities using encryption keys.
2. The **Operational phase** represents the state when the secrets are available, and trustworthy communication links established, allowing applications' information to be securely transmitted.
3. The **Post-operational phase** involves secrets not being anymore in normal use, since they are deactivated or in a compromised state. Despite being obsolete, secrets are not yet destroyed but may be archived.
4. The **Destroyed phase** imposes secrets being no longer available since they have been permanently deleted from a TND entity.

Beside KMP, the strength of the encryption algorithm is a critical factor for securing TND. Vital elements for choosing the encryption algorithm is also its performance and the computational effort required for encrypting and decrypting exchanged data. In that sense, symmetric encryption is favored over asymmetric encryption since the symmetric encryption achieves the same level of trustworthiness using shorter encryption keys. For example, in 2015 acknowledged secure key lengths were 128 bits for AES symmetric and 2048 bit for RSA asymmetric encryption algorithm [BD22]. Though the symmetric encryption offers better performances, due to the shorter encryption keys deployed on communication parties, Symmetric Encryption-based KMP (SEKMP) introduce several drawbacks regarding scalability for disseminating security material (e.g., encryption key tables, master encryption keys, or key derivation secrets) in a TND (cf. Section 2.3.2). Also, SEKMPs require the security material distribution before the realization of KMP procedures, following two approaches: (1) **Probabilistic key distribution** - having certain probability that two TND nodes will discover a common key during trust establishment, and (2) **Deterministic key distribution** - creating the encryption keys pool deterministically and with uniform distribution so that each two TND nodes share a mutual key [NLO15]. [Bar22]

Asymmetric encryption is useful since it relies on different encryption keys - public and private, for information encryption and decryption, respectively. In asymmetric encryption, the public key is known to all network entities, while the associated private key is securely stored in the network entity. Aside the from data decryption, the private key is commonly used for digitally signing the data, therefore providing data integrity and sender's authenticity [Sta16]. Network security solutions based on Asymmetric Encryption-based KMP (AEKMP) provide improved scalability compared to the SEKMP (cf. Section 2.3.2). Widely-accepted procedures for AEKMP are Diffie-Hellman [Sta16] and Elgamal [Sta11], as well as the standardized protocols like TLS [Res18] and DTLS [RM12]. However, compared to the SEKMPs, the major drawback of ASKMPs is high computational cost and energy consumption. The crucial assumption for secure AEKMP is the distributed public keys' authenticity. Public key authenticity is guaranteed by a *Trust Anchor (TA)* - a TND entity that is implicitly trusted by all other TND entities. Implicit trust for TA is achieved through the offline distribution of its public key, by storing it in the TND entities' memory through, for example, the operating system or installed applications [Bar22]. [Sta16]

2.3.2 *Trust Models*

TNDs employ various organization strategies concerning the management of security material and trust establishment, depending on the scenario for which they are used. These strategies are split into three main models: (1) **Direct trust**, (2) **Web of Trust**, and (3) **Hierarchical trust** [Hu16].

In the **Direct trust model**, trustor and trustee exchange their secrets and establish trust relationships in a manner that is immediately convincing to them, i.e. without the need for the trusted third-party. A common way of achieving direct trust is the security material distribution and configuration before the network deployment so that entities can establish trust during the system's run-time without further external assistance. The Direct trust model in a network mostly involves the distribution of symmetric key pair during manufacturing or system integration, enabling data confidentiality and integrity, as well as entity authenticity during system's runtime. Moreover, this approach utilizes the efficiency of the symmetric encryption compared to the asymmetric one [KBL18a; KBL18b]. However, this approach puts up a burden for the trust maintenance through the key management during the system's runtime, since any secret revocation and renewal has to be propagated to every entity in the network. Moreover, for the network of *N* nodes, pairwise shared symmetric keys require storage space of total *N*(N-1)/2* keys, of which *(N-1)* keys are stored in every node, making it unsuitable for large-scale networks [KBL18b]. [Hu16]

Another approach is based on the digital certificates whitelisting, i.e. instead with symmetric keys, entities are equipped with the digital certificates and certificate whitelists. During manufacturing, each entity obtains its digital certificate signed by a trusted authority, which ensures unforgeable proof of the entity's authenticity during the system's runtime. Moreover, the complementary whitelist is stored in each node containing references to the digital certificates of all trustable network peers [FF14]. While this approach reduces the overall number of keys in the network, compared to the method using symmetric key pairs, the whitelists maintenance is impractical for dynamic and large networks [Hu16].

The **Web of Trust model** utilizes trust propagation between peers for improving trust management in dynamic environments. In the Web of Trust model, the trustor can establish trust relationships with a trustee if the credentials of the trustee are trusted by another entity that has already established trust relationships with the trustor [GZV11]. For that purpose, the trustor relies on trust propagation techniques like direct trust inference, recommendation, reputation, and similarity-based trust prediction [SSD09]. This model is employed in Pretty Good Privacy [Cal+07], where individual nodes maintain a list of trusted public keys in a key ring. When a public key is inserted, its legitimacy is assigned by other peers in the network by marking the added key as trusted or not trusted. While Web of Trust improves support for networks' dynamics (peers entering or leaving the TND) compared to the direct trust model, it still does not offer efficient capabilities for non-static networks, such as vehicular or ad-hoc sensor network. Namely, Web of Trust introduces several drawbacks to the TND: (1) it disables new previously unknown peers to join TND, (2) it allows identity spoofing through unmanaged trust management infrastructure, and (3) it does not offer efficient means for the trust revocation since it is propagated to all TND nodes [Hu16].

The **Hierarchical trust model** involves the presence of one or more intermediary entities - TA in the TND, which serve to establish trust relationships between the communicating nodes that do not have previous trust relationships. In the case of the presence of multiple TAs, they form the hierarchical infrastructure for trust provisioning, named Trust Center Infrastructure (TCI) [Hu16].

To gain the ability to act as a trusted intermediary, TAs have to establish trust relationships with all the nodes in the TND, for which they afterward will provision trust relationships. For the trust establishment, TAs rely on the exchange of encryption keys before or during the system's runtime, whereas the methods mentioned above and trust models can be employed.

Kerberos [Neu+05] represents a protocol that utilizes TA to establish hierarchical trust in the Local Area Network or within a corporate network [Sta11]. Initially, Kerberos was developed to allow users to access distributed services in the network (e.g., printing or file-transfer) from their workstations [Sta16]. To achieve that, Kerberos introduces TA named KDC, which handles the symmetric keys for users and services and issues them encrypted permissions (Tickets) for accessing services. Compared to the Direct trust and Web of Trust models, trust maintenance is simplified through centralization in KDC, which requires handling of N symmetric keys for N users or services [Sta11]. Moreover, trust revocation and renewal through KDC is straightforward, since the keys and whitelists are managed only within KDC [Hu16]. However, KDC is also the critical security component representing SPoF, since compromising KDC would lead to failure of trust management in the whole TND and, therefore, lack of integrity and availability in the entire network [Hu16; Sta11].

PKI [Cho+03] is another example of the hierarchical trust model. Compared to Kerberos, which is centralized through the mandatory presence of KDC, PKI forms a hierarchy of TAs, called Certificate Authority (CA). Moreover, PKI relies on the public key encryption for trust establishment through the use of digital certificates [Coo+08a]. Digital certificates are representing encryption-verifiable credentials, proving the authenticity of a TND entity. Due to that, TND nodes can establish trustworthy communication links without the simultaneous trust mediation by the CA, making the PKI less susceptible to attacks on availability and trust integrity [Hu16]. Digital certificates' life cycle represents the core of the PKI [Hu16]. The lifecycle starts with certificates issuance by a CA, which acknowledges the integrity and authenticity of the certificate enclosed identity information by digitally signing it using CA's private key. Through the digital signature, CA vouches for the trustworthiness of the entity holding the certificate, which is afterwards verified by other TND nodes that are establishing trust relationships with the certificate holder. Nonetheless, the certificate's lifecycle ends with revocation, which occurs after a predefined time or if the certificate's trustworthiness is compromised by an error or malicious event in the TND [Sta11]. Presence of multiples CAs enables PKI's transitive trust properties, which are established through CAs' digital signatures, offering multilevel PKI hierarchies. Multilevel PKI allows the certificate holders to act as a CA and issue further certificates to other TND entities at a lower hierarchical level in PKI [Hu16]. Through that, PKI scales concerning the number of managed TND nodes, since the certificates management is leveraged between multiple CAs. [KPS02]

2.3.3 *Identity Management*

Besides encryption and trust management, the establishment of trustworthy communication links depends on the authentication of network peers. That involves IdM in the TND and enforcement of the mutual authentication procedures between entities [Mah+10a]. IdM enables representing and recognizing network entities as digital identities [22k; CLC15]. Thus, it includes processes and policies involved in managing the identity lifecycle properties: value, type, and optional metadata (attributes) for the identities known in a particular domain [22c; 22b]. IdM represents an essential block of IT systems since they heavily rely on digital identities for building security mechanisms and management features [Zhu+17].

IdM is provided through the Identity Management System (IdMS) [22m], which enables *"mechanism comprising of policies, procedures, technology, and other resources for maintaining identity information, including associated metadata"* [22c]. The main features offered by IdMS as per [Mah+10b; LC16] are:

1. Representation of entity through identity and its organization in namespaces (domains);
2. Storing entities' relevant information (e.g., names, credentials) as attributes;
3. Providing access to identity and attributes through standard interfaces.

The listed features ensure scalability for extensible entity structures through a resilient, distributed, and high-performance infrastructure [Mah+10b; Zhu+17]. Therefore, the IdM is accomplished through four operations that are offered by Identity Provider (IdP): (1) Identity Registration, (2) Identity Revocation, (3) Identity Update, and (4) Identity Lookup [Zhu+17] [CLC15].

Providing required IdM operations rely on the assignment of a digital identity to network entities through the information set that uniquely identifies the entity in the specific context [Zhu+17]. The information set must contain all the data that allow unique entity identification and authentication. Therefore, digital identity is defined as a three-tuple: identifier, credentials (private key or other authentication data), and attributes (additional entity information) [CLC15; Zhu+17]. Identifiers and attributes need to provide intuitive means for recognizing network entities.

Designing an identifier construction scheme involves defining semantic levels of the entities' identity. This corresponds to the ability to obtain descriptive information (attributes, meta-data) on the network entity, representing any relevant characteristic or property of an entity in a particular domain [22c]. For example, the Domain Name System (DNS) system utilizes Uniform Resource Locator (URL) addresses providing sufficient means to reason the offered services located on that platform since they are envisioned as a human-readable representation of IP addresses. For that reason, DNS addresses are considered as "strong semantics," while IP addresses as "weak semantics" since they do not provide additional attributes on a network entity [Hu16]. Nonetheless, to support unique identification, as well as "strong semantics" of network entities, identifiers are often split between so-called core identifier (reference identifier), representing unique value for identification purposes, and additional attributes that provide a semantic description of the entity [RZL13; 22c]. The core identifier is the domain-wide unique identifier that is intended to remain constant during the entity's lifecycle in the TND [22c]. A further aspect of building identifiers and attributes relates to naming convention, i.e. core identity and attribute names representation format. The naming convention provides a structure for binding information about the IdM namespace with the specific information concerning digital identity (e.g., user core identifier, attributes or role). A suitable naming convention is beneficial for the overall identity management by declaring interconnections between different entities in a structured manner. Moreover, query language provisioning for the IdM identity lookup operation depends heavily on the identity naming convention. A common way for a structured naming convention is flattening structure into text separated with dots. For example, DNS [Moc87] imposes *name.administrative domain* (e.g., .com or .org) format for namespaces and addresses. Furthermore, IPv6 [DH17] and MAC [EA13] network addresses utilize the same message format by having each text segment as a placeholder for separate information on a network entity. Using this message convention, a complex structure of identity namespaces is simplified as a single text without hindering identity lookup operation.

Another aspect of IdM is the possibility to federate digital identities' information between various platforms, allowing users to reuse their identities. In the isolated IdM approach, service providers provide IdM services to users, therefore neglecting the possibility to reuse digital identity information for accessing other service providers [Zhu+17]. Federated IdM (FIdM) is based on an agreement between two or more TNDs, specifying the exchange of digital identity information and the management of cross-domain identification procedures [Sha+18; Zhu+17]. Provisioning FIdM requires the establishment of trust relationships between service providers and their TNDs. Once the different TNDs trust each other, the exchanged information's authenticity is ensured. Establishing trust between TNDs most commonly employs a direct trust model (cf. Section 2.3.2), where encryption keys and digital certificates are exchanged using offline key exchange between integrating TNDs. Thereby, both symmetric and asymmetric encryption keys are used. For example, the OAuth2 IdM extension - OpenID Connect relies on the symmetric encryption exchange [JH12], whereas the Liberty Alliance Project[14] solution relies on the asymmetric encryption and digitally signed SAML tokens [CMJ15] for ensuring trust between TNDs. Moreover, Shibboleth also utilizes the asymmetric encryption to ensure TNDs' trustworthiness and relies on the hierarchical trust model through X.509 certificates [Mor+04].

2.3.4 *IoT Trustworthy Networks' Implementation*

IoT devices and IoT networks are vulnerable in many aspects [Bor14; ČH18]. For that reason, trust provisioning and maintenance put up numerous challenges. The biggest hurdle in the IoT system's security represents low-computational, battery-powered Thing's resources, incapable of supporting sufficient cryptographic primitives required for trustworthy communication [Hu16]. Moreover, a variety of Things in the IoT network, with different computational and storage capabilities, hinder the application of unique IdM and security protocols in the IoT TNDs.

Currently, Things identifiers are built based on manufacturer and hardware information (e.g., RFID or serial number) and network addresses - IP and MAC. However, this information offers low-security levels and weak authentication procedures, allowing spoofing and identity theft. Moreover, network addresses do not describe the general Thing's identity, but its identity in the context of the connected network interface. Therefore, building Thing's identity based on Thing's properties is set as a goal for IoT IdM. Thing's properties are divided into four categories: Inheritance, Association, Knowledge, and Context. [LC16]

The **Inheritance category** answers the question, ”What a Thing is?”. This question is answered by the information that Things hold memorized as part of hardware or software. Physical Unclonable Functions [MV10], as well as watermarks, security printing, and holograms are representatives for the inheri-

14 http://www.projectliberty.org/, last access May 2, 2022

tance category. However, inheritance category information is not as flexible as further categories since it relies on fixed information installed by the hardware manufacturer. Currently, this category is used in the system with high-security requirements. The **Association category** relies on associations that the Thing has developed during its lifecycle, like communication with the particular network gateway, smartphone or smart wearable. Furthermore, association based identity can be correlated with the Thing's owner information to improve the uniqueness and trustworthiness of the Thing's identity. The **Knowledge category** incorporates information the Thing can learn during installation and its runtime. Representative is the International Mobile Station Equipment Identity. Moreover, the measurement that Thing has previously collected represent knowledge that can be used to construct Thing's identity. The **Context category** is based on Thing's behavior. Behavior properties are gathered from IoT monitoring and analysis modules and mapped to identities through cryptographic operations. While this category offers adaptability for Things' dynamic and heterogeneous nature, collected context information is imprecise and prone to noise. Therefore, it is hard to detect the Thing's behavior during state transitions, as well as to derive Thing's identity from its behavior. [Zhu+17]

IdM deployment, accompanied by identities and secrets dissemination strategies, is categorized in multiple categories [22c; Zhu+17]: (1) Centralized, (2) Distributed, (3) User-centric, (4) Isolated, and (5) Federated. Each category employs different trust models (cf. Section 2.3.2) for building TN between multiple IdPs, end-users, and services. Centralized IdMs rely on third-party identity providers, which are most commonly deployed separated from application services [Zhu+17], as in, for example, Kerberos protocol [Neu+05]. The centralized IdM approach has its limitations, mostly concerning the scalability and availability in the highly-distributed IoT environments [Sha+18]. Distributed IdM systems allow multiple deployments of multiple IdP within the TND, enabling various points of control in the network for IdM functions. X.509 digital certificate-based PKI [Ada+05] is a representative of the distributed IdM. This results in a higher management effort required to ensure the consistent identity registry state but also improves the IdM's availability, fault-tolerance, and scalability.

IdM in IoT systems has to provide scalable support for a variety of devices in highly dynamic networks. Using a centralized approach, where the IdM logic is mainly located in one central entity, TND nodes need to establish a trust relationship just with that entity that will handle identification and authentication for them. However, due to the distributed nature of IoT (cf. Section 2.2), centralization of IdM hinders the possibility for Things collaboration and establishment of the edge intelligence, where Things are communicating directly, without the external entity's intervention. As a result, the distributed approach for IdM offers greater scalability and flexibility for IoT systems. Moreover, the IoT system can be built using multiple TNDs, with each one having a separate IdM system, therefore creating more efficient and trustworthy local IdM services. [RZL13]

Concerning Things' trust management, the main building blocks are isolation and cryptography. The degree of application of isolation and cryptography mainly depends on the Things' computational and communication capabilities and is implemented both in the Things' hardware and software modules. The Hardware Security Module (HSM) provides a higher level of trust, especially for storing cryptographic keys and operations, which is achieved through tamper-resistant hardware modules optimized for security operations (e.g., key generation and storage, data encryption and decryption) [Hu16]. The main representative of HSM is the Trusted Platform Module (TPM), implemented on the device as a chip separated from the primary device's CPU. The application of HSM dramatically extends the Thing's battery life. However, HSM fails to meet the requirement for encryption keys management and update procedures and adjusts to the encryption key formats or algorithms that were defined after HSM is deployment [Hu16]. Compared to that, software security components are implemented as an isolated area on the device's platform that is specialized for security functions, named Trusted Execution Environment, offering a higher level of security compared to the security operation in the central platform. Software security modules also include software stores, providing cryptographically-protected storage for encryption keys, tokens, and passwords [Hu16]. [22b]

Besides the IdM and cryptographic material protection, TN in IoT requires protection of communication links between Things, Edge Devices, and DCs. For that, information exchange between communicating entities should employ [22b]:

- Explicit endpoint communication policies,
- Strong mutual authentication between endpoints,
- AC rules enforcement on exchanged information, and
- Cryptographic mechanisms to ensure confidentiality, integrity, and freshness of exchanged information.

With that goal, communication nodes need to establish secure communication links through mutual authentication and encryption keys exchange, which will be used to encrypt and decrypt exchanged data. Secure transfer of secrets for Things named key distribution schemes or KMPs are categorized according to the applied encryption algorithms into two areas [NLO15; KBL18a], following SEKMP and AEKMP concepts described in Section 2.3.1.

Utilization of AEKMP and SEKMP provides strong security guarantees for ensuring data confidentiality. Still, these KMPs rely on algorithms that have a high demand on the storage and computational resources [KBL18a; Had+19; RZL13]. Things' constraints concerning computing and memory capacity, as well as energy resources, result in the Things' inability to apply traditional security protocols and algorithms, i.e. Internet Protocol Security (IPSec), Secure Sockets Layer (SSL), or Rivest–Shamir–Adleman (RSA), that involve strong encryption schemes in IoT systems [LC16; Bon+12a]. To address these challenges, several lightweight cryptography methods are analyzed and reported [McK+17; KKV12]. Still, even lightweight encryption schemes cannot be adopted by highly-constrained Things (e.g., Radio-Frequency Identification (RFID) Tags or low-energy Bluetooth devices) [Sha+18], minimizing the possibility of finding a universal solution that would ensure E2E security in IoT systems. This affects also the KMP and IdM protocols in IoT, since they are mostly based on the computationally-demanding AEKMPs (e.g., RSA, Elliptic Curve Digital Signature Algorithm (ECDSA), or Diffie-Hellman) [ARC18; RZL13; Sha+18; GMS15]. Therefore, the management of the security configuration by Things would lead to an increase of Things' complexity to support higher computation and energy demands [Bor14]. Moreover, Things would be required to handle the security configuration (keys, identities, credentials) for all communicating nodes in the IoT network, leading to a further increase in their complexity. For those reasons, such an approach is not viable since it would significantly increase the IoT system's costs and energy consumption.

IoT networks are often equipped with computationally-rich devices like FNs, Smart Home Gateways or Routers. These devices can support traditional security protocols and algorithms and achieve a higher level of information security than Things [Sha+18]. Since these devices are deployed in the IoT system's edge layer, they can bridge the gap between Things and remote DCs concerning security settings management in local IoT networks [Sha+18; Bon+12a]. Relatively stable relationship between edge devices and Things can be used to establish trust between them, allowing offloading computationally-demanding tasks like data encryption, key generation, and intrusion detection, as well as algorithms and protocols: Datagram Transport Layer Security (DTLS) and Elliptic Curve

Cryptography (ECC) [GMS15], ZigBee [Fan+17], and IdM [Zhu+17], from low-capable Things to more powerful edge devices [Sha+18; GMS15]. Through that, edge devices can take over the trust management from CC services, enabling security protocols and management schemes to scale to the larger number of connected devices [Had+19], as well as reduce the communication latency and improve performance of security features in IoT systems [Sha+18]. Moreover, the edge layer becomes the central point for security management in IoT networks, simplifying the security materials management through the centralized identities and credentials distribution [Sha+18]. However, offloading security mechanisms to the edge layer has to be accompanied by trust relationships' management between Things and edge devices, requiring Things' trust into the edge device to which the Thing offloads computational tasks. Therefore, a Thing has to securely connect to the edge layer using the lightweight security protocol that the Thing supports [Sha+18].

2.4 Access Control

AC represents an essential part of the network security, focusing on providing the data confidentiality CIA principle. It is defined as one of the basic security mechanisms in ITU-T Recommendation X.800 [22l] as *"the prevention of unauthorized use of a resource, including the prevention of the use of a resource in an unauthorized manner"*. RFC 4949 provides multiple definitions for AC [Shi07]:

1. *"The prevention of unauthorized use of a resource, including the prevention of the use of a resource in an unauthorized manner"*;
2. *"A process by which the use of system resources is regulated according to a security policy and is permitted only by authorized entities (users, programs, processes, or other systems) according to that policy"*;
3. *"Limitations on interactions between subjects and objects in an information system"*.

For a general understanding of AC definitions, it is vital to define a security policy from [Shi07] as *"a set of policy rules (or principles) that direct how a system (or an organization) provides security services to protect sensitive and critical system resources"*. Fundamentally all of the definitions mentioned above point to limiting the access to the system resources and therefore protecting the system information. In order to achieve that goal, several procedures must be applied, divided into three main categories of AC, also known under the acronym AAA [Laa+00; Shi07]:

1. *Authentication* is the process of verifying a claim that a system entity or a system resource has a certain attribute value;
2. *Authorization* represents an approval that is granted to a system entity to access a system resource;
3. *Accounting* is the property of a system or system resource that ensures that the actions of a system entity may be traced uniquely to that entity, which can then be held responsible for its actions.

By observing the definitions of each AAA point term, a conclusion can be made that AC enforces following steps: (i) identification of a system's entity and proving its authenticity, (ii) validation of access rights (permissions, privileges) for the authenticated entity, concerning predefined system's security policies and thereby deny or grant access to the resource, and (iii) tracking and auditing performed actions in the system so that each action can be uniquely traced to its origin. From the system architecture standpoint, Access Control Center (ACC) has been defined in [Shi07] as ”*a computer that maintains a database defining the security policy for an access control service, and that acts as a server for clients requesting access control decisions*”. To enable AC services, ACC is deployed within an administrative domain, defined as *”a collection of end systems, intermediate systems, and subnetworks operated by a single organization or administrative authority.”* [HK89]. The design of ACC is directly impacted by the AC management, which involves the distribution of the security policies, described through the security attributes (e.g., credentials or access rights). The selection of security attributes depends on a used AC model (cf. Section 2.4.2) for the particular system.

Having one of the leading roles in the network security area, AC services can be implemented using various mechanisms (firewalls, subnet-masking, application-level authorization) on different layers of ISO/OSI basic reference model [22d]. AC support on ISO/OSI layers is presented in Table 2.4.

In the Sections 2.4.1 to 2.4.5 different AC aspects and the challenges and requirements for the application of AC on IoT systems will be examined.

2.4.1 *Authentication*

Authentication refers to the assurance that the communicating entity is the one that it claims to be [Sta16]. Therefore, authentication should validate and prove the authenticity of the entity, defined as *”the property of being genuine and able to be verified and be trusted”* [Shi07]. Depending on the security object specific to a security service, authentication is classified as follows [Hu16]:

Table 2.4: ISO/OSI reference model's layers [22l]

Layer	AC support
Application	Application protocols and/or application processes can provide application-oriented AC facilities.
Presentation	No support.
Session	No support.
Transport	AC is employed on a E2E transport connection basis.
Network	AC mechanisms can be used to control access to sub-networks by relay entities and access to end systems.
Link	No support.
Physical	No support.

1. *Message authentication* is the assurance of message integrity and the origin that is preserving a message from unauthorized alteration;
2. *Entity authentication* is the assurance of the entity's identity.

According to the [Shi07], an authentication[15] process consists of two steps:

1. Identification involves presenting the claimed attribute (identity) value to the authentication subsystem.
2. Verification involves presenting or generating authentication information that acts as evidence to prove the binding between the attribute and the identity for which it is claimed.

During the identification step, the entity provides its digital identity [22k] to the ACC by sending through the use of the identifier, containing system-wide unique information on the entity, distinguishing it from all others [Shi07]. For that, digital identities have to be designed using a standard naming scheme, addressing specific characteristics of the network entities (e.g., Things or end-users). Moreover, the complete lifecycle of the network entities has to be followed by a proper digital identity management and maintenance, which is provided through the operations of IdMS (cf. Section 2.3.3) [22m].

15 Terminology concerning authentication can be misleading, because terms authentication and entity authentication are often used as synonyms. For the sake of simplicity and better understanding of the definitions, these two terms will also be used as synonyms in the rest of this dissertation.

The verification step requires presenting authentic information about an entity. This verification can be categorized under the three factors, namely authentication mechanisms classes [NN06]:

1. Something the authenticating entity **has**, i.e. hardware token, smart card, or a digital certificate;
2. Something the authenticating entity **knows**, i.e. password and PIN code;
3. Something the authenticating entity **is**, i.e. entity's physical characteristic - fingerprint or iris.

It is important to point out that identity verification can employ classes of the multiple authentication mechanisms (e.g., a password + iris scan or SMS TAN code), which is defined as multi-factor authentication [Sta11]. The output of the successful authentication is a session token or encryption key, which represents the user's digital identity for a particular period of time, limited with token's validity. This token serves for confidentiality and authorization purposes (cf. Section 2.4.2) once a user requests access to a resource.

2.4.2 *Authorization*

Authorization is *"a process for granting approval to a system entity to access a system resource."* [Shi07]. Therefore, authorization represents a set of procedures that determine if and under which conditions the access to the particular resource can be granted to an entity based on the presented credentials. A credential represents security information which is previously obtained from ACC in a two-folded manner, i.e. through (1) *Authentication* - Successful identity proving and obtaining session token (cf. Section 2.4.1) or (2) *Accounting* - acquisition of revenue that authorizes a user for the access to the particular service [NN06]. Once provided with a credential, authorization performs the matching of information from the credential with previously configured access privileges (rights, policies) for the requested resource. Access privileges reflect the security policy (cf. Section 2.4) for the administrative domain.

AC or Authorization[16] modeling defines the approach for mapping security policies to access privileges. Therefore, the AC model bridges the gap between the high-level security policies and low-level mechanisms by defining the means

16 In literature the terms "AC modeling" and "Authorization modeling" are used as synonyms and, thus, usually the acronym "AC" stands for both. For the sake of simplicity and a better understanding of the definitions in this dissertation, here the term "AC modeling" is applied.

of how authorization rules should be applied to protect resources. Critical aspects for choosing an appropriate AC model for the particular administrative domain are: (1) a method for describing the digital identity (profile) of the user - a set of attributes for the user description and (2) an access rights management policy (distributed or centralized). With those aspects in mind, various AC models are researched and published. [Oua+17]

With regard to the access rights management policy **Mandatory Access Control (MAC)** and **Discretionary Access Control (DAC)** models are defined. MAC model enforces a strict management policy for access privileges by restricting the assignment of access rights exclusively to a group or individual that has the authority to manage the domain's access rights [And11]. Thereby, the owner of the resource or service is not allowed to manage resource's access rights. This approach reflects rather centralized access rights management and is widely used in high-risk areas, i.e. military and government. In contrast, DAC enables the owner a full control of the access rights for a resource [CMF12]. Through that, the owner can share the resources with other users in the administrative domain. Due to its distributed nature and minimization of the centralized access rights management, various systems utilize DAC, such as social networks and the UNIX file system.

The approach for describing the user's digital identity initiated the design of multiple AC models. The **Role-Based Access Control (RBAC) model** [San98] describes the user's profile solely through his role in the administrative domain (e.g., administrator or student). RBAC offers simple access rights management for a group of users, enabling the access to a user for the particular resource for a user without creating a new role in most cases. However, RBAC negatively impacts the access policy granularity, causing the creation of additional roles if access to a resource should be granted or denied for an individual user. For providing access rights for a single user without the overhead of a new role, the **Identity-Based Access Control (IBAC) model** [GQ18] is used. Using IBAC, access rights can be granted using the user's identifier (e.g., email, passport number, or social security number). Although this allows fine-grained access rights definitions, management for a big number of users in such an administrative domain can become cumbersome and time-demanding.

Overcoming the drawbacks of RBAC and IBAC concerning the access rights management, the **Attribute-Based Access Control (ABAC) model** [Hu+22] bases user's profile description on an *attribute* - any generic property of a user, such as role, age, or nationality. With that, ABAC is capable of provisioning fine-granular access rights definition for single users, but also a group of users based on any users' common attribute. The **Capability-Based Access Control (CAPBAC) model** [GPR13] partially follows ABAC's approach by enabling the user profile description using multiple properties. However, instead of relying on the user attributes, CAPBAC bases access rights definitions on the capabilities, representing allowed operations the user can execute in the administrative domain. In contrast, **Lattice-Based Access Control (LATBAC) model** [San93] follows a different approach than the previous models. Foundation of LATBAC are *labels* - generic administrative properties defining the security levels that a resource has and which user needs to match to authorize an access request. Through this, LATBAC allows generic, fine-grained access rights management, since that labels are not strictly bound to any user's or resource's property.

2.4.3 *Accounting*

According to the [Laa+00], the purpose of the accounting is to *"generate any relevant accounting information regarding the authorization decision and the associated, authorized session (if any) that represents the ongoing consumption of those services or resources"*. Referring to this definition, accounting serves to trace the access request towards the services or resources in a system, and for that relies on the results from the authentication and authorization phases, in need of authorized session and authorization decision information, respectively. Based on outputs of the accounting, a variety of application categories are defined [NN06]:

1. *Auditing* is the process for the verification of an invoice issued by the service providers for the previous system usage;
2. *Cost allocation* represents analysis of costs with regard to usage portions of the specific service;
3. *Trend analysis* enables forecasting future usage and required capacity for the particular service.

As can be seen from the definition for the accounting and the main auditing application areas, even though it is a part of AC, accounting does not represent a security mechanism, such as authentication and authorization. The primary purposes of accounting are the system's usage tracking and generating metrics for further system development and planning. Nonetheless, positioning the accounting in the AAA architecture, as well as deploying accounting services in the ACC, represents killing two birds with one stone, since all of the information required for the accounting are present in ACC, making the data analysis for accounting purposes simpler.

Once the accounting information is collected, it is forwarded to the other components in the system. Forwarding occurs using two models: **pooling model** and **event-driven model**. The **Pooling model** relies on the forwarding of accounting data at regular, periodical intervals, usually by the accounting manager. For the sake of preventing the data loss, the period between two data collections has to be shorter than the maximum time the data is stored in ACC. In the **event-driven model**, the ACC publishes the data to the accounting servers, either once a particular event in the system occurs, or a periodical interval in ACC expires. In this manner, depending on the storage capabilities of the ACC, as well as the type of event (e.g., monthly usage collection or fraud detection), ACC publishes accounting data in batches or as a single accounting event. [NN06]

2.4.4 *Multi-domain Access Control*

The AAA mechanisms described in Sections 2.4.1 to 2.4.3 provide AC services for a single computer system, that is a single administrative domain (referred to as domain in the remainder of this section). These mechanisms differ from domain to domain, each enforcing its security policies, such as credentials distribution, password policies, and authorization attributes. Security policies and accompanying security information (e.g, username or password) partially have to be maintained by the end-user, which presents an additional burden. This reflects mostly the users' need to memorize and apply different passwords once visiting various domains. In order to reduce the maintenance overhead for end-users' security information, Multi-Domain Access Control Systems (MDACS) have been introduced. These systems represent a federation [22k] of domains that interconnect their ACCs in order to provide access to the resources without the need for the user to re-authenticate each time once accessing the resource in a different domain. MDACS can be categorized based on the AAA procedures: (1) Authentication - FIdM systems and (2) Authorization frameworks.

Even though MDACS differ in their purposes, they share a typical computing architecture pattern. The central entity in MDACS is the Authoritative Server, i.e. the ACC exposing its internal AC services to the other federated domains, through the exchange of authorization messages. The exposure of AC service has to be followed with security mechanisms in order to provide a successful and trustworthy operation of MDACS. These security mechanisms ensure a certain level of trust between federated domains. Establishment and maintenance of trust relations between federated domains require cryptographic procedures on authorization messages exchanges from and to the Authoritative Server, with the goal of proving the authenticity of an authorization message and the federated domains. PKI and digital signatures ensure the required authenticity since federated rely on X.509 certificates to confirm their identity. Moreover, using the private keys pairing X.509 certificates, authoritative servers digitally sign a message and append the digital signature to the exchanged authorization message. The digital signature is afterwards used by other communicating parties to validate the authenticity and integrity of the authorization message. With the goal of a standardized provisioning approach for authorization messages format, along with digital signatures, XML-based Security Assertion Markup Language (SAML) [22i], and JSON Web Token (JWT) [JBS15] standards have been introduced.

FIdM systems enable user authentication across multiple federated security domains, namely Single Sign-On (SSO). SSO mechanism enables users to perform a single authentication procedure and obtain access to all domains within FIdM [RR12]. FIdM systems introduce several benefits to the system actors [Cha09]:

1. Enabling users with SSO capability, allowing them to access services from different service providers without the need to re-authenticate;
2. Offloading service providers' costs for managing users' digital identities and security information;
3. Providing greater system scalability, allowing service providers to offer services to a greater number of users;
4. Allowing identity providers to maintain close relationships with the users and offer them additional services.

The first approach towards FIdM systems was based on X.509 certificates, where each user would obtain a private and a public key pair, as well as an X.509 certificate containing a globally unique identifier for a user. This user identifier would be recognized by all federated domains, allowing user authentication based on a digital signature. The biggest constraint of such a system is privacy concerns since everyone in the federation would know everyone else's identifier. [Cha09]. Due to that reason, further solutions have been developed, which rely on distributed storage of user's security information in federated domains. This security information is exchanged between federated domains only in scenarios when the user requests resource access in other domains, with explicit consent from the user. Hence, private information sharing remains under the user's control, leading to the minimization of privacy concerns.

Shibboleth is a representative of such an approach. Initially established by Internet2 through Middleware Initiative (I2MI)[17], Shibboleth relied on the PKI and SAML standards for building inter-domain trust. Furthermore, Shibboleth requires integrating third-parties to implement an authoritative server, that is, Shibboleth IdP, which issues SAML-based session tokens to the user upon successful authentication. These session tokens are afterwards presented to the Shibboleth Service Provider during authorization. Since the session token does not carry all user security information, in case of insufficient information during the authorization, a Service Provider is due to request further user's security information from IdP, which are obtained with the user's consent. Thereby, the user keeps control of the private information. [Mor+04]

Multi-domain authorization protocols were designed to resolve the privacy issues related to large-scale Internet applications [Oua+17]. The primary motivation for the multi-domain authorization protocols was granting the access to private user's information for third-party applications. Thereby, the biggest challenge was the enforcement of the possibility for the user authentication exclusively at the authoritative server, which protects user information. The OAuth2.0 protocol [Har12] has been developed with that goal. In the OAuth2.0 protocol, IdP is called authoritative server. An authoritative server manages user security information and issues JWT [JBS15] based session tokens upon successful authentication. Session tokens are then used by the third-party application for the local session management and for accessing the user information in the Resource Server, to whom the access is controlled by IdP. Trust between IdP and third-party application is established by exchanging and configuring security keys, as well as a unique application identifier for application authentication purposes at the OAuth2.0 authoritative server. Since its definition the OAuth2.0 protocol has been adopted worldwide, including major service providers such as Facebook, Microsoft, and Google [Oua+17].

17 `https://www.internet2.edu/products-services/trust-identity/shibboleth/`, last access May 2, 2022

2.4.5 *Access Control in IoT*

Even though AC is a well-researched area of network security, it is mainly applied in CC architectures, where servers are grouped and connected in a DC. Therefore, traditional AC solutions are designed and developed around a single ACC and multiple clients (third-party server, applications), which are authenticated and authorized within ACC. In contrast, IoT introduces numerous servers in the security domain, since not just CC servers, but also Things, EDC, or FN can be servers. Due to that, several application challenge for AC to support AAA mechanisms in IoT systems emerged, as summarized in Table 2.5. [Oua+17]

Table 2.5: AC justifications causing IoT requirements [Oua+17]

AC challenge	Description
Heterogeneity	Building collaborative AC environment for the devices of various vendors.
Data management and Privacy	Provisioning users with a control over their private information.
AC policies management	Deployment and maintenance of access policies in numerous administrative domains.
Reliability	Continuity of AC service in spite of network failures.
Trust	Establishment and maintenance of trust relationships between different administrative domains.
Fine-granularity	Expressiveness of syntax used for access rights definition.
C-A	Application of contextual factors as inputs for access rights validation.
Scalability	Extensibility in size, structure, and number of potential users and Things.
Lightweight	Support for low-computational solutions due to devices' resources constraints.
Cost	Economical aspects for building and maintaining IoT AC system.

2.5 Context-Awareness

C-A represents the core feature of computer systems developed in the areas of pervasive and ubiquitous computing, as well as in the intelligent environments, providing support for adapting systems behavior based on the gathered context data [Per+14a]. The terms C-A and Context-Aware System (C-AS) have been defined by Dey et al. [Abo+99a] as *"a system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task"*. Context-aware computer system is any computer system able to use the context, i.e. analyze and interpret insights on their environment's current state and adapt provided services and functionalities without the explicit user intervention. This minimizes user's effort required for the computer system management, leading to enhanced usability and effectiveness of the deployed services. [BDR07a; AAC16]

Context interpretation and analysis refer to the transformation of context information in a way that special knowledge derived from measured sensor data is linked and stored along with the sensor data [BDR07a; Per+14a]. A clear distinction between the raw sensor data and context information is explained by Sanchez et al. [San+06], declaring the sensor data as *"unprocessed and retrieved directly from the data source, such as sensors"* while context information is *"generated by processing raw sensor data. Further, it is checked for consistency and metadata is added"*.

The variety of the deployed sensors, as well as accessible context information, create limitless scenarios for establishing C-AS. Initially, location information is considered as by far the most used context information [BDR07a]. However, the increasing popularity of the intelligent environment and IoT produced various C-ASs, accompanied by architectures, models, and context analysis techniques [Per+14a].

The popularity of the C-ASs required standardizing mechanisms for defining, identifying, quantifying, analyzing the context in an IT system. Even though most people intuitively understand the term *context*, they find it hard to outline the concrete definition for context. Synonyms for context that can be found in dictionaries are circumstance, situation, phase, position, attitude, terms, status, surroundings, location, dependence, etc. [Per+14a]. Due to the broad meaning the context can have, many researchers tried defining context through the information that can be collected in a C-AS:

- Ryan et al. [RPM99] identified context as the user's location, environment, time, and identity;
- Dey [Dey98] identified context as the user's emotional state and focus of attention, location, orientation, time, and other devices and users in the environment;
- Brown [Bro96] identified context as the elements of the user's environment which the computer knows about.

Still, the definitions mentioned above are considered as too specific, and that they offer no consensus on the definition of context [Per+14a; AAC16; Abo+99a]. For that reason, Dey et al. [Abo+99a] provided a definition that identifies context in a broader sense and is the most acknowledged until now [AAC16]. Namely, Dey defined context as ”*any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*”.

As mentioned before, integrating C-A reduces the required users' effort for IT system management. The IT system should understand not only simple contexts, such as user location or room's temperature, but also complex context scenarios like the social context and user behavior. Complex context scenarios contain a complex relation between context information, human activity, and human behavior, which is somewhat hard to detect and predict. Therefore, the expectations on C-ASs can differ from their capabilities, i.e. features that they offer. Satisfying these expectations is vital for adopting C-ASs since the user sometimes has a better understanding of what is happening, and if the system produces unexpected behavior, the user can reject and abandon the system. [AAC16]

It is considered that the context arises from an activity [Gre01; Dou04; AAC16]. Thereby, *”activity comprises a subject (the person or group doing the activity), an object (the need or desire that motivates the activity), and operations (the way an activity is carried out). Artifacts and environment are seen as entities that mediate activity”* [Gre01]. That implies that the context information can be produced and maintained through each activity in the system, thus being highly dynamic. However, collected context information does not have to be relevant for each activity [Dou04]. This increases the complexity of C-ASs since it becomes very hard to derive inter-dependencies between various contexts and how they are affected by an activity, leading to building the commonly disconnected context analysis tools that are focused on solving specific, isolated issues [AAC16].

The main problem of building C-ASs is that software developers must identify the context they need to program. To tackle that challenge, developers have to: *"(A) enumerate the set of contextual states that may exist; (B) know what information could accurately determine a contextual state within that set; (C) state what appropriate action should be taken in a particular state"* [Gre01]. To resolve the challenges, various building blocks (e.g., system architectures, models, or features) for designing C-ASs are researched.

Upcoming sections present these building blocks: (1) main concepts for building C-ASs and their main features in Section 2.5.1, (2) categorization of the context information in Section 2.5.2, (3) approaches for building C-ASs in Section 2.5.3, and (4) lifecycle of C-ASs in Section 2.5.4. Finally, the application of context in IoT systems is presented in Section 2.5.5.

2.5.1 *Context-Aware Systems Fundamentals*

The process of building C-ASs, i.e. embedding C-A into computer systems, requires tackling diverse challenges that can be dramatically different depending on the integrated context type, aimed context-aware features, etc. [AAC16]. Still, C-ASs fundamentally adhere to several approaches for their supported features classification, and approaches for building them.

Dey et al. [Abo+99a] groups the features a C-AS can offer as follows: (1) Presentation of information to the stakeholders, (2) Tagging context to information, (3) Active or passive service execution, and (4) Active or passive service configuration. **Presentation** allows usage of context to decide on what information and services should be presented to the user. This enables the possibility to present information based on context, such as time, location, or user behavior. **Tagging** the context information enables simple retrieval of this information in the future, allowing faster and simpler processing of the context information and optimizing C-AS's performances. **Execution** of services allows C-ASs to execute an action depending on the environment's context, with or without the user's intervention. This is a critical feature for IoT since it allows automatic control of smart devices (e.g., turning on air condition so that the room temperature is adequate once the user arrives or turning off the lights once the user leaves the room). **Configuration** relies on the context-based personalization, allowing a C-AS to learn user preferences, offering configuration options for the user-optimized system behavior. This reduces the C-AS development complexity and enables system configuration for the inexperienced users. Both execution and configuration rely on the previously mentioned C-AS interaction options and can be passive, which requires user's direct involvement in the actions, or active, where the C-AS undertakes actions autonomously. Still, these features do not have to be completely active or passive in a C-AS. Hybrid approaches applying both passive and active interactions are possible. [Per+14a; AAC16]

Utilizing C-AS features involves interaction between the C-AS and a user. For that purpose, Barkhuus and Dey [BD03] defined the following interaction levels:

1. Personalization enables users to manually configure their preferences, likes, and expectations to the C-AS;
2. Passive C-A defines a C-AS that continually monitors the environment and offers context-based options to the user to perform actions;
3. Active C-A defines a C-AS that constantly monitors the environment and performs actions autonomously upon some activity in the environment.

Approaches for building C-AS have been examined by Hu et al. [HIR08], resulting in the definition of the three main models for building C-AS. The **No application-level** context model envisions that all the context processing tasks (e.g., context acquisition, pre-processing, storing, and reasoning) are performed within the application boundaries and developed as part of the application. The **Implicit context model** proposes that applications use libraries, frameworks, and toolkits to perform the context processing tasks. This model provides standard guidelines for building C-ASs that make such systems' development faster and easier. However, the context is still hard bound to the application. The **Explicit context model** enforces application to use a context management infrastructure or middleware software. Therefore, the context processing tasks are executed separately, outside the application boundaries. This enables separate development and deployment of the context management and application, allowing more effortless extension of the application and context model.

2.5.2 *Context Information Handling*

C-ASs' features and interaction options described in Section 2.5.1 rely on an IT system's capability to comprehend and manage context information from its environment and make it available to system applications [Dey+01]. Furthermore, a C-AS has to separate the context acquisition and exposure to the application and users so that an application can use the contextual information without having to know the details on the context acquisition [Bet+10].

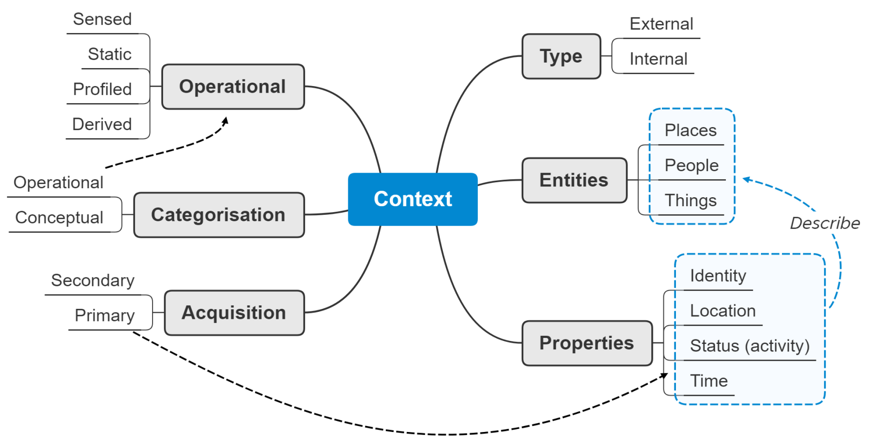


Figure 2.4: Context information categories

Successful context information management is based upon the context model definition [BDR07a]. Henricksen et al. [HI04] define a context model as follows: *"a context model identifies a concrete subset of the context that is realistically attainable from sensors, applications and users and able to be exploited in the execution of the task. The context model that is employed by a given context-aware application is usually explicitly specified by the application developer, but may evolve over time"*. In the same publication, Henricksen et al. identify context attribute as the primary building block of the context model and define it as *"an element of the context model describing the context. A context attribute has an identifier, a type and a value, and optionally a collection of properties describing specific characteristics"*. Baldauf et al. [BDR07a] propose the following context attribute properties set for providing a more comprehensive context attribute description: timestamp, context source, and confidence.

Acquired context information varies through time and can depend on the complexity of the designed context model. For that reason, context information is error-prone and does not represent a single source of truth [BFA05]. Context information is extended through the context attributes describing Quality of Context, which incorporates a parameter set that expresses the quality of the context information's requirements and properties [Per+14a]. The Quality of Context parameter set is based on three parameters: (1) context data validity, (2) context data precision, and (3) context data up-to-dateness [Bel+12].

An essential aspect of the context information management is the context identification, whose analysis is useful to the system's applications and users. Context identification elements are analyzed in this dissertation's scope using the publications [BDR07a; Per+14a; AAC16; Abo+99a; HI04; SAW94; DAS01; Hof+03; PB03; Gus02]. The resulting list of these aspects is presented in Figure 2.4.

Context separation by acquisition type is introduced by Abowd et al. and described as primary and secondary [DAS01]. Primary context represents any information that can be retrieved without performing any sensor data aggregation operations or using existing context information - for example, location, identity, time, and activity. Secondary context is any information that can be computed using the primary context through the sensor data aggregation or data retrieval from context management services.

Context is categorized based on two schemes: operational and conceptual [AAC16]. Conceptual category enables understanding interdependencies between different contexts. Operational deals with the techniques of the context data acquisition. Henricksen [HI04] categorized context into four categories using operational categorization:

1. Sensed context represents context acquisition directly from the sensors;
2. Static context manages information which will not change over time;
3. Profiled context handles information that changes over time with a low frequency, such as once per month;
4. Derived context represents the information computed using primary context.

Based on the conceptual categorization, Schilit et al. [SAW94] introduced context identification based on three questions: (1) **Where you are?** (2) **Who you are with?** and (3) **What resources are nearby?** [Per+14a]. These questions lead to establishing entities that are useful for detecting context [DAS01]: (i) places (buildings, room), (ii) Things (sensors, actuators, computer components), and (iii) people (individuals, groups). Each entity can be defined through multiple properties separated into four categories: identity, location, status (activity), and time [DAS01; Per+14a].

Lastly, identified context type is classified through two dimensions. The first dimension is called *internal* [PB03; Gus02] or *logical* [Hof+03]. This dimension identifies the context specified by the user and its interactions in the C-AS, such as the user's tasks, work context, emotional state, or behavior. The second dimension is external [PB03; Gus02] or physical [Hof+03] and refers to the context measured by sensors, such as temperature, humidity, or location. [BDR07a]

2.5.3 *Building Context-Aware Systems*

The multitude of special requirements and conditions, such as sensors location, users number or available computational resources dictate the C-ASs' software architecture. For that reason, there is no universally accepted approach for designing the C-ASs' architecture [AAC16]. However, various architectural elements are common to the context-aware architectures. [BDR07a]

The main driving factor for C-AS design is the context acquisition approach [BDR07a], which is derived from the context acquisition type - primary and secondary (cf. Section 2.5.2). Chen [Che03] classifies context acquisition as:

1. Direct sensor access enables context information collection directly from sensor devices (primary context);
2. Middleware infrastructure (secondary context) encapsulates low-level sensing details and allows context information collection through programming interfaces. It separates context acquisition and usage, thereby enabling easier development of the context-based application by reusing context information management middleware [AAC16].
3. Context server (secondary context) allows access to the context information from multiple concurrent context-based applications through remotely-invokable services. Thus, computationally intensive context information management operations are leveraged to separate software components, which reduces the context-based applications' hardware requirement.

An architecture using a context server is currently the most used one for building a C-AS. Despite being highly centralized, with one or many centralized components, it overcomes the storage and computational constraints of small devices in a C-AS, which is significant for IoT [BDR07a]. Furthermore, decoupling the context server's architecture into multiple modules is highly important [AAC16]. A Context server's generic architecture that involves multiple modules encapsulating functionalities is proposed by Baldauf et al. [BDR07a] and is divided into layers: (1) Sensors, (2) Raw data retrieval, (3) Storage/Management, (4) Preprocessing, and (5) Application. Through these layers, context management functionalities like context collection, analysis, and application management are decoupled, allowing simpler context data sources extensions and new context applications registration [BDR07a].

Independent from the chosen architecture, several design principles apply to the C-AS design process. Table 2.6 presents design principles related to C-ASs collected from publications [AAC16; MLA10; Ram+07; BTC08].

Table 2.6: Design principles for C-ASs

Design principle	Description
Architecture layers and components	Meaningful separation of the functionalities into layers and components.
Scalability and extensibility	Dynamic addition and removal of components.
Reliability	Ensuring continuous services delivery along with tolerance for errors and faults.
Application programming interface (API)	Functionalities exposure through comprehensive, simple API.
Debugging mechanisms	Sufficient tools for debugging functionalities.
Automatic context lifecycle management	Ability to automatically detect available context entities (cf. Figure 2.4), their structure and derive context model based on this information.
Context model in-dependency	Storing context model separate from C-AS source code, enabling independent alteration of them.
Comprehensive context modeling	Context models covering managed context information can be easily extended for further use-cases.
Multi-model reasoning	The ability of the context-aware system to support different context models.
Mobility support	The ability for C-AS to be deployed on various devices, such as gateways, PCs, mobile phones, etc.
Share information (real-time and historic)	Ability to share context information between distributed components in C-AS.
Resource optimisation	Data structure and algorithm optimization to reduce demands on storage and energy, especially in IoT use-cases with a high number of devices, i.e. 50 billion.
Monitoring and event detection	Ability to detect an event in the system and undertake appropriate actions.

Finally, context information management deals with private users' information, such as their current activity, location, and behavioral patterns [BDR07a]. Furthermore, context analysis increases the security threats that aim at misusing context information [Per+14a]. For that reason, establishing sufficient security mechanisms to guarantee user's privacy is a critical point for C-ASs design [BDR07a]. Security and privacy must be protected in several architecture layers: sensor hardware layer, data communication layer, as well as context collection, modeling, and distribution layers [Per+14a]. This requires creating a trustworthy environment (cf. Section 2.3) for the context information management by applying TN, AC (cf. Section 2.4.5), secured data transmission, and other security mechanisms.

2.5.4 *Context Management Lifecycle*

Context information management occurs in multiple phases. These phases are referred to as the context lifecycle, which clarifies the operations application in each phase [Per+14a]. Phases in the context lifecycle encapsulate the following processes: context acquisition, data processing, context reasoning, and distribution to context-based applications [BDR07a; DAS01; BTC08; AAC16]. Based on these phases, Perera et al. [Per+14a] presented the framework for defining the context lifecycle that contains four phases: collection, modeling, reasoning, and distribution, where each phases depends on the results of the previous one.

First, context information **collection** needs to be performed. As context information is collected from multiple distributed sources, the context information's quality, authenticity, and structure are challenging to achieve [AAC16]. To tackle these challenges and provide a systematic approach for context information acquisition, Perera et al. [Per+14a] propose five factors related to the acquisition process that must be considered when designing a C-AS. Table 2.7 presents these five factors, possible options, and their descriptions for each factor. **Context modeling** supports the context information expression and management. It enables the process of translating collected context information into usable, structured values [AAC16]. Context modeling contains two steps that are defined in [Per+14a]. In the first step, the context information model is defined in terms of context attributes, properties, and relationships with other context information in a C-AS. In the second step, the modeled context information is validated against the C-AS requirements. Requirements for context information models are defined in [Bet+10; PRL09] as: (i) Represent any context information kind, reflecting the real-world entities and dependencies between them; (ii) Uniquely identify the contextual information, context, and entities; (iii) Simplicity, reusability, expandability, and ability to use the information at runtime; (iv) Validate collected data and encode its uncertainty. Context models researched in several publications [Bet+10; SL04; CK00; BDR07a] are (1) Key-Value, (2) Markup scheme, (3) Graphical, (4) Object-oriented, (5) Logic-based, and (6) Ontology-based.

Table 2.7: Context acquisition factors

Factors	Option	Description
Responsibility	Pull	The data is retrieved from the sensors with a request.
	Push	The sensor pushes data to the software component which is responsible to acquiring sensor data periodically.
Frequency	Instant	Context-related events occur instantly and do not span across certain period.
	Interval	Context-related events span a certain period .
Source	Sensor	Context is acquired directly from sensor.
	Middleware	Context is acquired through middleware infrastructure.
	Context server	Context is acquired from other context storages (databases, web services) through API.
Sensor Type	Physical	Generates sensor data by itself.
	Virtual	Does not generate data and can retrieve data from many sources publishing sensor data.
	Logical	Combination of physical and virtual sensor type.
Acquisition Process	Sense	Data is sensed through sensors.
	Derive	Context information is generated through sensor data processing.
	Manually provided	User provides context information manually (e.g., configuration or preferences).

Key-Value models represent the simplest context models. In these models, context information is structured as key-value pairs in different formats like text and binary files. Due to their simplicity, key-value models cannot support attaching meta-information or context attributes. Also, they are not scalable and capable of storing complex data structures. **Markup scheme models (Tagged Encoding)** rely on markup tags based on techniques like XML to store context information. This enables efficient context information retrieval and context validation through markup scheme definitions. However, markup languages do not provide expressive capabilities required by the context reasoning, which hinders the context reasoning techniques' performance. **Graphical models** allow a higher expressiveness degree than key-value and markup models by enabling the relationships between context information to be captured in the context model. They incorporate standard languages like Unified Modeling Language (UML) [18] to model context information while storing low-level context information using database languages (e.g., SQL and NoSQL). This allows storing a vast amount of information in a database, but also may demand execution of very complex and non-performant SQL queries for complex context information handling. **Object-oriented models** are applied to model context information using object-oriented programming concepts like class hierarchies and their relationships. This allows a more straightforward integration of the context model into the C-ASs since most high-level programming languages are based on object-oriented programming principles. Due to this, object-oriented models are mostly implemented in the C-AS's code, not externally using other languages like XML and UML. Therefore, interoperability of such models is reduced, as well as context model validation capabilities. **Logic-based models** rely on facts, expressions, and rules to represent context information. Through that, context information relationships are described as natural patterns that occur between them, which allows better performing derivation and aggregation of high-level context information using the low-level context. This allows the usage of logic-based models by non-technical users. However, logic-based models suffer from a lack of standardization, which hinders their interoperability and re-usability in modern C-ASs. **Ontology-based models** structure context information into ontologies [SBF98] using the standardized semantic technologies, i.e. Resource Description Framework (RDF) or Web Ontology Language (OWL). This allows the description of complex relationships between context information. However, context information retrieval can be time-consuming and computationally

18 https://www.omg.org/spec/UML/2.0, last access May 2, 2022

demanding when the amount of collected data is increased. Still, ontology-based models are supported by many development tools and reasoning engines and offer a common understanding of the context information structure between people or software systems [Wan+04]. [Per+14a]

As can be noticed, each group of context information models has its strengths and drawbacks. Still, despite their complexity concerning data structuring and information retrieval performances, ontologies are acknowledged by many surveys as the preferred mechanisms for managing and modeling context [Per+14a; AAC16; SL04]. Finally, context models are not exclusive and can be integrated into hybrid approaches to minimize their deficiencies [Bet+10].

Context reasoning represents a method of deriving new context information based on the available context [Per+14a]. Through the reasoning, context information is better understood and processed so that the context management system obtains a better overview of the C-AS's state. The need for context reasoning emerged due to the two properties of primary context: imperfection and uncertainty. Context reasoning models can be divided into three phases [NF]:

1. Context pre-processing cleans the collected sensor data;
2. Sensor data fusion combines sensor data from multiple sensors and produces more accurate, dependable, and complete context information;
3. Context inference generates high-level context information based on the lower-level context information.

Context reasoning techniques are classified into six categories: supervised learning, unsupervised learning, rules, fuzzy logic, ontological reasoning, and probabilistic reasoning [AAC16]. Still, each technique has its own strengths and weaknesses. For that reason, C-ASs simultaneously apply multiple context reasoning techniques to reduce their weaknesses and accomplish better results concerning the derivation of context information. [Per+14a]

The **context distribution** phase provides methods for delivering context information to its consumers, such as middleware infrastructure, context servers, and client applications. For that purpose, two approaches are envisioned. The first one is **query-based**, meaning that consumers request context information via a query that is handled by a context management system, which produces results sent as a response. The **subscription (publish/subscribe)** approach allows context consumers to subscribe with a context management system to context information. Afterward, the context management system publishes the context information to the consumer based on the previous subscription. [Per+14a]

2.5.5 *Context-Awareness in IoT*

IoT services run in highly dynamic, mobile environments, with numerous sensors and actuators sensing data from the environment. The exploitation of the collected data through context analysis and applying C-ASs features (cf. Section 2.5.1) significantly enriches IoT services intelligence and establishment of the smart spaces [Bor14]. Intelligent IoT services can execute actions in the IoT environment autonomously and proactively, communicate collaboratively with other platforms to improve IoT system functionality, annotate sensed data to optimize processing performances, etc. [AIM10]. This vision of IoT is emphasized in the documents of the European Commission: *"Things having identities and virtual personalities operating in smart spaces using intelligent interfaces to connect and communicate within social, environmental, and user contexts"*[19].

However, IoT systems diversity, categorized through several IoT application domains, hinders the development of the general context model for IoT [Per+14a]. Thus, IoT-based C-ASs development tends to be more application domain-specific [RG18; EJK19]. Nevertheless, the context-aware IoT solution's common requirement is the application of semantic technologies for building context-model. This leads to the optimized context information processing, which is crucial characteristic for C-ASs deployment on resource-constrained devices [Per+14a]. Smart Home environments benefit the most from the C-ASs since they target improving users' quality of life by adapting the IoT environment state (room temperature, air humidity). To achieve that, Smart Home systems rely on context information like system time, sensor measurements, and user location [Kim+16]. Despite the numerous benefits for Smart Homes, C-ASs tend to be vendor-oriented, resulting in a lack of common context middleware support [Kim+16; VZN12]. Furthermore, numerous sensors in Smart Cities allow comprehensive context analysis to optimize IoT services in cities [Per+14b]. Application of C-A in vehicular networks utilizes context information collected by the vehicles and processes it to propose drivers safer, less congested, and more efficient driving routes [Wan+14].

Except for the optimization and smartification of the IoT services, context is often considered as the promising approach for enhancing IoT security. To achieve that, a set of contextual attributes can be used to derive the IoT environments state and reconfigure security policies to protect information from unauthorized access, modification, destruction, or disclosure [Das+16; TTČ17]. Firstly, context-aware

19 `https://docbox.etsi.org/erm/Open/CERP%2020080609-10/Internet-of-Things_in_2020_EC-EPoSS_Workshop_Report_2008_v1-1.pdf`, last access May 2, 2022

security approaches are seen as complementary techniques to cryptographic solutions, which is highly beneficial for resource-constrained Things [Had+19]. Secondly, in the scope of AC, the context has multiple implications. Firstly, context analysis can be used for authenticating users by analyzing their behavior patterns, which reduces the user's effort required for authentication using passwords or biometrics [Hul+05]. Moreover, the extension of security policies used for authorization through the context information can enable the adaptation of access rules based on the current IoT system's state [CBE00; Kay+20b].

2.6 Summary & Findings

In this chapter an overview and state of the art with regard to building FC-based, trustworthy, context-aware IoT systems is provided. Firstly, IoT systems have been described, along with the services they are enabling, possible applications in IT systems categorized through IoT application domains, and the present challenges for building IoT solutions. The critical challenge is the Things' lack of computational resources, which enforces the IoT services deployment on remote CC servers. However, as analyzed in Section 2.2.1, CC imposes its drawbacks that hinder the innovations in IoT. To overcome these hurdles, novel computing paradigms focusing on the computational resource deployment at the IoT networks' edge are being introduced. FC stands out as a promising candidate for extending CC and offering computation resources close to the Things through the utilization of the existing hardware in IoT networks (e.g., gateways, routers, and smartphones).

When it comes to securing IoT systems, as documented in Section 2.3, establishing trust relationships between communicating entities plays a critical role, which is achieved through the mutual authentication and unique identification in IoT networks. Thereby, the application of resource-demanding cryptographic operations for encryption keys distribution hinders the utilization of the traditional key exchange protocols on resource-constrained Things. Furthermore, IoT networks' dynamicity implies performance and scalability requirements to which the trust management models fail to comply. To overcome these issues, approaches for establishing lightweight, scalable trust management infrastructure for IoT are heavily researched. Research strategies in this area focus on the introduction of novel, lightweight protocols and the reduction of computational requirements in the existing protocols so that they can be applied to a variety of IoT devices.

AC is the second factor that assures the protection of information in IoT. As documented in Section 2.4, AC has been thoroughly researched and widely applied in computer systems across several ISO/OSI layers through various mechanisms and protocols, such as firewalls, VLAN, NAT, SAML, and OAuth2. However, these mechanisms are often centralized in DCs, making them not scalable and unable to support novel computing distribution approaches such as FC or EC. IoT is considered as a highly distributed system, where service providers are not located merely in DC but deployed in the whole network as Things. Therefore, further research and development of the mechanisms for AC distribution are required (cf. Section 2.4.4), fulfilling the requirements of IoT systems and leading to low-latency, E2E security in the IoT world. [Oua+17]

Finally, context information management and utilization have vital importance for the smartification of IoT environments, that is, the establishment of intelligent, autonomous IoT services and the improvement of IoT systems' security. Despite the common approaches for C-ASs design and development that are discussed in Section 2.5, the diversity and complexity of IoT systems put up challenges for the design and development of context-aware IoT solutions, mainly concerning context modeling and establishing standard architecture for the context management. Current C-ASs application in IoT solutions focuses on enabling novel, user-friendly, appealing services, leading to the broader adoption of IoT systems. Besides these services, integration of C-A into IoT security introduces research opportunities, resulting in improved IoT data protection through intelligent, adaptive security policies which require less users' effort for their management and maintenance. [CBE00; Kay+20b; Kay+20a]

3

Design Decisions

As described in Chapter 2, IoT networks incorporate a variety of physical devices and provide innovative services in the several IoT application domains. The number of connected devices and their low-computational storage and computational capabilities put up a burden for the produced data management and processing. CC has been massively used for offloading resource-demanding tasks from Things and offering applications and services for end-users. Yet, the drawbacks of CC partially hinder the future development of the IoT system (cf. Section 2.2.1). For that reason, novel, decentralized computing platforms like MEC, MCC, EC, and FC have been introduced to overcome the CC downsides.

Moreover, securing data information in IoT networks represents another challenge for IoT system architects, mainly due to Things' incapability of performing highly computationally demanding cryptographic operations. These limitations imply offloading security services to the remote DCs, therefore making Things more insecure and less trustworthy for the end-users. Through decentralized computing platforms, security mechanisms (e.g., AC and mutual authentication) can be deployed closer to Things, creating a trustworthy IoT execution environment.

The information collected in IoT systems provide important insights that can be used for securing IoT networks (cf. Section 2.5). This fact gains more significance since operators of IoT systems are not necessarily technical persons. Therefore, the IoT environment context-based security mechanisms automation can further improve IoT systems' trustworthiness and leverage the security management tasks of end-users.

Shifting security services from DC and bringing them closer to the Things as well as making them more automatic is the central goal of this dissertation. For that reason, this chapter presents the analysis of the requirements and challenges, as well as the design steps to achieve the previously mentioned goals. Firstly, Section 3.1 lists and analyzes the requirements and features of significance defined by a standard FC architecture to deploy IoT security service on FNs. On that basis, Sections 3.2 and 3.3 provide solution design analysis and steps for the security services deployment within the Cloud-Fog-Things continuum, i.e. TN, and AC, respectively. Automation through C-A AC is described in Section 3.4. Finally, design decisions and specifications from the Sections 3.1 to 3.4 are summarized, providing insights for the implementation described in Chapter 4.

3.1 Solution Requirements & Goals

Offloading computational tasks from Things to external devices (e.g., DC, EDC, and FN) pawed the way for the adoption of IoT. Remote and centralized DCs offer significant computing power but introduce performance and efficiency drawbacks to the IoT, such as network latency, high bandwidth consumption, and processing latency. In contrast, FC extends CC as a computing platform deployed closer to the Things. Still, FC does not neglect the presence of CC, but is defined as its extension and the intermediation layer between CC and Things, offering a more efficient and better-performing computing platform in cases when it is required. FC processing, storage, and networking capabilities reside in FNs. FNs are highly-virtualized entities, meaning they can be deployed as software components on any device with enough resources that can be dedicated to FN execution. FNs' hierarchical organization depends on the complexity and computational requirements of the hosted IoT service, as well as on FN's computational capabilities. FNs with more resources are usually positioned higher in the hierarchy. However, the nature of the hosted IoT service is the determining factor for FN's hierarchical position.

To deploy software-based IoT security services using FC, a set of general FC characteristics and requirements defined in [22g; Bel+19; Bon+12b] have been analyzed. In the analysis (cf. Figure 3.1), FC characteristics and requirements are matched against (1) the goals of this dissertation and (2) the general FC architecture [22a]. Based on the analysis, the solution design decisions for FN deployment are derived and presented in Section 3.1.2. Moreover, the requirements for the design of FC-based IoT security services application support are collected and further analyzed in Sections 3.2 to 3.4.

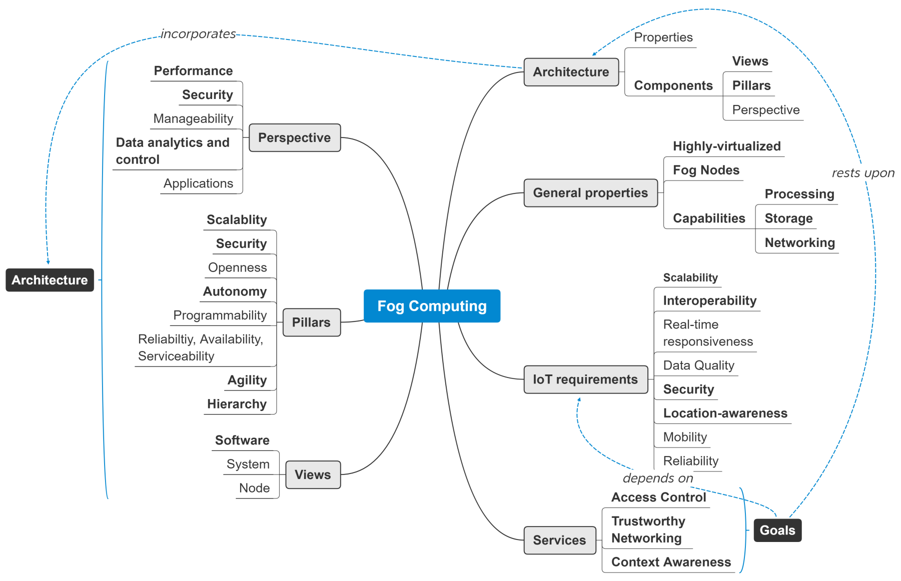


Figure 3.1: Fog Computing characteristics and requirements

3.1.1 *Solution Goals & General Requirements*

As mentioned in Chapter 1, the final goal and central contribution of this dissertation is the provisioning of IoT security services, namely C-A AC and TN in IoT systems. To overcome the CC drawbacks collected in Section 2.2.1 and offer better performing and more efficient security services, FC is used as a platform for services deployment. This section provides a brief description of the security services, as well as general requirements for their deployment based on identified CC drawbacks and general FC requirements presented in Section 3.1.2.

AC prevents unauthorized access to the IoT services, i.e. Things and the measurements previously collected by Things [221]. To achieve that, ACC evaluates each resource access based on the previously defined access rights. Since it is involved in each IoT services invocation, to reduce the network communication overhead, ACC should be deployed close to the IoT service providers. The AC service for IoT systems described in this dissertation is designed and implemented to (i) reduce network latency, (ii) improve efficiency and scalability of AC in IoT, and (iii) enable support for intelligent, C-A AC by incorporating the sensed and analyzed IoT environment's state. These challenges have been thoroughly analyzed and mapped to the requirements in Section 3.3.

TN deals with the secured communication links establishment between network entities within a TND [22o]. Secured communications are often established through intermediary trusted entities that build TCI around one or multiple TAs. Each TA requires sufficient storage and computational resources to provide cryptographic protection for the trust relationship management within TND. In the IoT, computational and storage resources are mostly located within DCs, often distant from Things. DC distance makes the trust management more complex since TCI covers a great physical area. Moreover, having a vital support for cryptographic operation remote from Things makes them more vulnerable to network attacks. Decentralization and distribution of TCI and separation of TNDs presented in this dissertation provides several benefits for the TN in IoT systems: (i) localization and isolation of TNDs based on the FN computing and storage capabilities, as well as (ii) a more straightforward and more efficient trust management. A more in-depth analysis of these goals, as well as solution design steps in the area of trustworthy networking, are provided in Section 3.2.

C-A deals with the collection, analysis, and distribution of context information in an IoT environment. Obtained context information is used for the optimization of the existing and introduction of novel IoT services. While having FC-based security services deployment as the central point in this dissertation, C-A is applied to improve AC services. In order to achieve this, the following aspects of C-A integration into AC mechanisms are designed and developed: (i) creating a generic data model for integrating context information into AC, (ii) an API for exchanging context information between C-A agents and AC components, and (iii) the extension of authorization procedures to incorporate context information. Further details concerning these aspects are presented in Section 3.4.

Deployment and execution of the above-mentioned security services must comply with the FC architecture [22a]. Furthermore, solution design decisions and their effects on the CC drawbacks declared in Section 2.2.1 are presented in the following paragraphs.

Deployment of AC and TN services on the FNs simplifies the management of security services and enables the creation of TND in the local network. Since locally deployed security services are functional without the presence of the remote DCs, the invocation of a remote security service is not required, therefore unpredictable network latency introduced through communication with remote Cloud Servers is reduced. Moreover, public networks (Internet) bandwidth consumption is minimized. Also, security service provisioning will not depend on the network quality, since its operability does not involve the constant invocation of the remote DC.

Deployment of FC-based C-A agents creates the possibility to sense and analyze information from the local IoT environment, deriving its state through context analysis. Obtained context information are afterwards used for automatic management and optimization of IoT services. In the scope of this dissertation, IoT context analysis will be used to improve the operability of the earlier mentioned security service. By having locally deployed C-A agents, IoT environment state analysis is handled locally, without the intermediation of Cloud Server, minimizing network and processing latency that occurs once performing the context analysis in the remote DC.

3.1.2 *Fog Computing Deployment Requirements*

Deployment of the AC and TN security services, as well as C-A agents through FC infrastructure, requires application of various FC requirements. In the scope of this dissertation, FC requirements from OFRA [22a] and other publications like [Bon+12b] and [Bel+19], have been analyzed. Requirements of interest have been listed, categorized, explained and presented in Table 3.1. The presented requirements, however, are not merely related to the security services deployment model but also affect the design and implementation of the security services themselves. For that reason, several FC requirements will be satisfied in the scope of the system architecture and services deployment (cf. Section 3.1.3). Remaining requirements will be satisfied through the security services design, namely AC (cf. Section 3.3), TN (cf. Section 3.2) and C-A services (cf. Section 3.4).

3.1.3 *Security Services Deployment Model*

As per requirements listed in Table 3.1, hosting IoT services on FN requires a variety of requirements to be satisfied. These requirements impact the IoT services deployment model, as well as the design and implementation of the services themselves. This section provides design decisions and their justification related to the IoT services deployment model. Namely, requirements **FC 1**, **FC 2**, **FC 3**, **FC 4**, and **FC 5** (cf. Table 3.1) are taken into consideration for the deployment model and general system architecture design.

Table 3.1: Fog Computing deployment requirements

Label	Category	Title	Source	Explanation
FC 1	Architecture	Highly-virtualized deployment	[22a; Bon+12b]	Containerization and isolation of services execution environments on FNs.
FC 2	Architecture	Hierarchical organization	[22a; Bon+12b]	Logical hierarchy based on the complexity and computational requirements of an E2E IoT system and FNs.
FC 3	Performance	Low network latency	[22a; Bon+12b; Bel+19]	IoT systems network latency minimization, IoT services real-time responsiveness, and reduced traffic congestion.
FC 4	Scalability	Scalability of FC network	[22a; Bon+12b; Bel+19]	Single or multiple FNs resources extendability to support IoT service computational, storage and network requirements.
FC 5	Autonomy	CC services independence	[22a]	Service provisioning regardless of the connectivity to the Cloud Server.
FC 6	Agility	Location and Context-Awareness	[22a; Bon+12b; Bel+19]	Improve and optimize IoT system operability through context- and location-based IoT environment's state analysis.
FC 7	Openness	Interoperability	[22a; Bel+19]	Standardized way to describe information across multiple IoT platforms. Support for different sensors, FNs through software abstraction layers.
FC 8	Security	Data security and privacy	[22a; Bon+12b]	User control over securely-stored private data.
FC 9	Security	E2E security	[22a; Bel+19]	CIA provisioning in Cloud - Fog - Thing continuum.
FC 10	Security	Trustworthy FC infrastructure	[22a]	Building and managing trust relationships between Cloud Servers and FNs.
FC 11	Security	Identity Management	[22a]	Cryptographically protected Cloud - Fog - Thing IdMS.

Security services deployment through FNs involves the creation of a highly-virtualized environment for services execution and hosting of computational, storage, and networking capabilities. Virtualization introduces benefits for the IoT system through the of the new services without the necessity to install new hardware. Therefore, resource efficiency in IoT devices increases through the execution of additional services and reduction of devices' idle CPU cycles. Virtualization techniques offer a platform-independent, emulated execution environment - a virtual machine that can be deployed on a physical device (host). Through the virtual machine, the abstraction of computing, storage, and networking resources is provided to the users and services. Furthermore, virtual machines can be multi-tenant, meaning that they can execute concurrently on the same host. Multi-tenancy is achieved through isolation of operating system components like kernel, network stack, and filesystem. Beside virtual machines, a novel, lightweight virtualization approach, called containers, based on the kernel and filesystem sharing between virtual tenants on the same host, is being widely adopted these days. Despite shared kernel and filesystem, containers offer isolated namespaces, as well as resource (CPU, memory) usage restrictions.

Based on the IoT devices' characteristics concerning the computational resource constraints, the deployment of the aimed security services should not introduce significant computational and storage overhead. Still, multi-tenant service support on the same FN is mandatory, since it enables better efficiency of the FN. Moreover, hosting multiple services on the same FN increases the overall scalability of the IoT system through the increased number of the deployed services. Under the influence of these arguments, the security service developed in this dissertation will employ containers for their deployment, providing a highly-virtual service execution environment. Complete isolation between containers is unfortunately not present since they share operating system components, i.e. file system and kernel. However, partial isolation and operating system components sharing lead to more lightweight and more efficient IoT services deployment.

As already mentioned, network latency and bandwidth consumption are some of the most critical CC drawbacks. The introduction of FC and similar computing paradigms deal with these drawbacks. In the scope of this dissertation, reducing network latency is set as one of the FC requirements, namely **FC 3**. To satisfy that requirement, the developed security services are deployed where most IoT services are provided - at the edge of the network, along with Things. The closeness between FNs and Things allows for security services to be provided without the invocation of CC servers, hence reducing network traffic congestion in the public IP network.

FC is generally considered as a CC extension. For that reason, complete elimination of CC's application contradicts the FC definition, even though it would mean a maximal reduction of network bandwidth consumption for IoT services. Thus, in this dissertation's scope, CC services are employed for the initial container deployment, as well as their configuration and management. Locally deployed security services are periodically and asynchronously configured by the Cloud Server. This results in a regular monitoring and maintenance of the FC-based security services without the need to immediately contact Cloud Server for operations that occur in the local network, minimizing bandwidth consumption and improving the responsiveness of the security services.

Basing the solution design on both CC and FC introduces a hierarchical organization of software, that is software components. As per **FC 2**, the definition of components' hierarchy is affected by present FN computational resources, as well as the offered service's complexity and computational requirements. In the presented solution architecture, hierarchical organization is twofold (cf. Figure 3.2). The top-level component hierarchy follows the standard FC-based IoT architecture, separating services on three layers: (1) Cloud, (2) Fog, and (3) Thing. The second-level hierarchy is emerging on the Fog layer, that is, based on security service dependencies and complexity, but also services separation and isolated deployment via containers. Namely, TN and C-A services require fewer computational resources than AC service. Therefore they can be deployed on more computationally constrained FNs, positioning them lower in the components hierarchy.

The overall solution architecture is presented in Figure 3.2. In this section, a brief components overview and their inter-dependencies is provided, whilst particularities of each component are provided in the corresponding sections:

1. Section 3.2 for **Trustworthy Network Trust Anchor (TNTA)**, **Fog Trust Anchor (FTA)** and **Fog Trust Provider (FTP)**;
2. Section 3.3 for **Access Control Agents Management (ACAM)** and **Fog Access Control Agent (FACA)**;
3. Section 3.4 for **Context-Awareness services**.

ACAM and TNTA are deployed on the Cloud Server. Therefore they represent remote, central units for the management of other FC-based components. Constant availability of these components is not required during the system's runtime since FC-based components can provide the desired service without invoking CC-based methods. Still, CC-based management of the FC-based components involves periodic and event-based configuration for the required security services.

Namely, ACAM deploys the configuration for access policies definition through the listing of supported IoT devices functions (cf. Section 3.3.2). TNTA represents the root of PKI-based hierarchical trust, therefore publishes certificate updates to FNs, maintaining the trust relationships in the IoT system (cf. Section 3.2.2).

Security services for AC, TN, and C-A deployed on FNs are implemented through FC-based components: FACA, FTA and FTP, as well as C-A agents, respectively. Since FACA and FTA communicate directly with CC components and through that apply configuration and management procedures to other FC-based components, they are positioned higher in the second-level hierarchy. FTP and C-A agents are completely internal FC components, that can be deployed on multiple FNs, therefore improving overall scalability and robustness of the designed solution. Services provided by FC components are:

1. FACA supports user authentication and validation of access requests on IoT services;
2. FTA and FTP offer operations for creation and management of the local TND, based on PKI and direct trust model for constrained Things;
3. C-A agent is responsible for sensing and analysis of the IoT environment's state, as well as forwarding the information to FACA for automatic access rights adaptation.

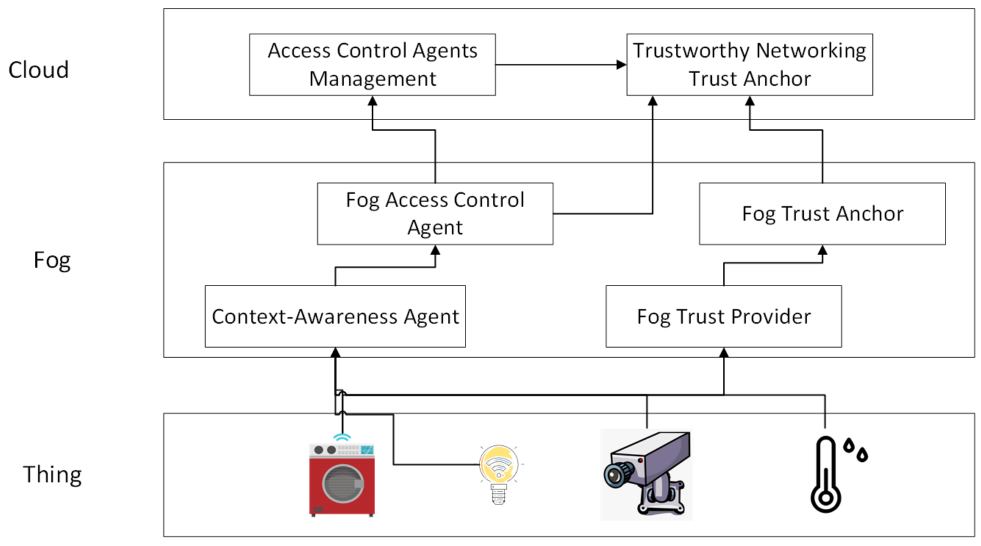


Figure 3.2: Solution architecture and components hierarchy

3.2 Trustworthy IoT Networks based on Fog Computing

TN aims at establishing a secure data exchange between entities in a TND. Secured data exchange involves protecting information from unauthorized access (confidentiality) and from altering it during the transmission (integrity). To achieve that, communicating nodes in TND perform mutual authentication. Based on the successful authentication, entities exchange secret encryption keys or credentials, which enable encryption and decryption of the transmitted information within the TND (cf. Section 2.3).

Trust establishment and maintenance involved several aspects concerning nodes identification, authentication, and encryption keys distribution and management within a TND presented in Figure 3.3. The realization of trustworthy communications during systems' runtime involves encryption material management through several phases, namely: (1) Pre-operational, (2) Operational, (3) Post-operational, and (4) Destroyed [BD22]. Supporting the mentioned aspects through the key management phases dictates the definition of several trust models, incorporating different cryptographic approaches for secure keys distribution, as depicted in Figure 3.3.

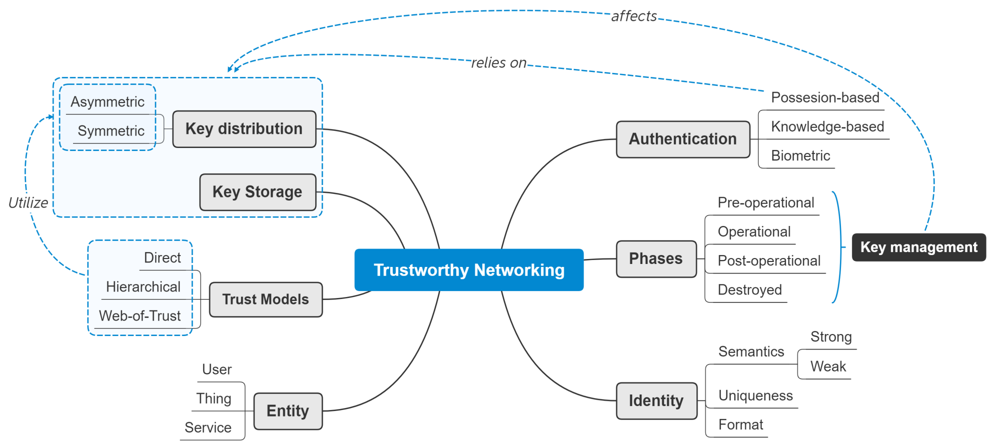


Figure 3.3: Trustworthy Networking aspects

The TN establishment in IoT systems introduces several challenges and requirements. The critical challenge is, by all means, the Things' computational and storage resource limitations, leading to Things' incapability of performing the required cryptographic operations. This incapability imposes hurdles for achieving sufficient means for the trustworthy exchange of information within a TND. This dissertation aims at using additional computing and storage capabilities of FNs, accompanied by deployed CC- and FC-based TN services: **TNTA**, **FTA** and **FTP**, to improve trustworthy communication within TNDs.

The TN services' design and implementation are influenced by two aspects. The first aspect consists of the FC deployment requirements (cf. Table 3.1) concerning TN: **FC 5**, **FC 7**, **FC 9**, **FC 10**, and **FC 11**. The second aspect contains the IoT system characteristics:

- **TC 1** Dynamic IoT networks,
- **TC 2** Heterogeneity of Things,
- **TC 3** Computationally constrained Things,
- **TC 4** Battery-powered Things,
- **TC 5** Low-latency requirements, and
- **TC 6** IoT services scalability due to the number of interconnected devices.

Enlisted FC deployment requirements and IoT system properties are analyzed for common topics, leading to the TN requirements definition (cf. Table 3.2).

IdM and KMP deployment within the proposed solution aims at providing the trustworthy FC infrastructure (**TN 1**), affecting FC-based IoT systems'characteristics and requirements: interoperability (**TN 4**), reliability (**TN 5**), and scalability (**TN 6**). IdM and KMP comply with each other since authentication procedures heavily rely on digital identities, which allows unification of provisioning and managing identities and secure secrets through common procedures. Defined IdM functions [LC16] are tightly coupled with KMP phases [BD22]:

- Register IdM function correlates to the KMP pre-operational phase;
- Identity lookup and update are provided during the KMP operational phase;
- KMP post-operational and destroyed phases rely on the identity revoke functionality.

Resolving defined general TN requirements involves the analysis of multiple TN aspects. Therefore, Section 3.2.1 proposes design decisions for IdM. Moreover, Sections 3.2.2 and 3.2.3 describe the approach for the trustworthy FC-based services deployment model and strategy analysis for ensuring E2E security in the FC-based IoT system, respectively.

Table 3.2: IoT Trustworthy Networking requirements

Label	Title	Relevant IoT characteristics	Design implications
TN 1	Trustworthy FC infrastructure (FC 10)	TC 1, TC 3, TC 4, TC 5, TC 6	Deployment of trustworthy FNs and software services and improving trustworthy communication between Things.
TN 2	Identity Management (FC 11)	TC 2, TC 6	Identification of deployed IoT devices and services through common identifier scheme.
TN 3	E2E security (FC 9)	TC 1, TC 3, TC 4, TC 5, TC 6	Utilizing FC to establish secure data transfer in Cloud - Fog - Things continuum.
TN 4	Interoperability (FC7)	TC 2	TN architecture design suiting the diversity of the deployed Things.
TN 5	Reliability (FC 5)	TC 1, TC 5, TC 6	Operability of TN services despite the failure of CC and FC services.
TN 6	Scalability (FC 4)	TC 1, TC 6	Achieving IoT system's low-latency and responsiveness along with TN provisioning.

3.2.1 *Identity Management*

IoT systems also heavily rely on IdM, most notably for Things identification and management. Compared to the traditional mainframe and CC systems, incorporation of IdM into IoT systems introduces additional challenges due to the heterogeneous, mobile nature of Things. Moreover, the computationally-constrained nature of IoT, as well as scalability requirements due to the number of interconnected Things, represent further hurdles for the IoT systems' IdM design [MPP13]. Thus, the ultimate goal for IoT IdMS is to provision universal IdM, facilitating interoperability between IoT application domains and platforms in both IoT services and Things naming, as well as identity-based authentication procedures and protocols.

Assigning and managing IoT identities provides a solid foundation for building cost and resource-effective IoT security solutions [LC16]. However, a universal approach for IoT IdM has not been defined yet, resulting in the current research on:

1. Defining identity in the IoT systems [LC16];
2. Establishing a common naming scheme to address the heterogeneous nature of Things [Sal+16b];
3. Introducing efficient management of identifiers and accompanying credentials [Zhu+17].

A major challenge for IdM system design is defining the unfalsified information set that can be used by the Things, IoT services and end-users, allowing them to mutually authenticate each other [Zhu+17]. End-users mostly rely on private, memorable information for their identity and credentials (username, e-mail, password). In contrast, IoT services are described through the software manufacturer and software version, secured through credentials in form of a digital signature. Consensus on Things identification has not yet reached the "one size fits all" solution (cf. Section 2.3.3).

Since each information category imposes benefits and drawbacks, the application of multiple categories for defining the Thing's core identity and attributes set occurs often. Based on the use case, information from different categories (cf. Section 2.3.4) are selected and used for a semantic-rich Things' identity. Since the category information can change during the Things lifecycle, it is beneficial to build core identity from static information. Therefore, attributes can posses static and dynamic Things' properties, which can be handled and obtained from IdP or sent joined with the core identity across the network. [LC16]

Designing and implementing a solution based on the IdM requirements (**TN 2**) in the scope of the dissertation requires the design of the identity naming scheme and management system with further characteristics: interoperability (**TN 4**), reliability (**TN 5**), and scalability (**TN 6**). The listed requirements will be resolved in the following aspects: naming scheme, IdM deployment model, and security credentials management (cf. Section 3.2.2).

The designed identity naming scheme aims at modeling the standard identity format for three entity types in the IoT environment: end-user, deployed services, and Things. Enabling the interoperability of the IdM solutions requires following the structured, dot-delimited naming convention described in Section 2.3.3. Nevertheless, this naming convention and hierarchy-based deployment model allows simple interdependencies replication between deployed services, improving an identifiers' semantic. Integrated into core identifiers, the naming scheme enables efficient lookup function within IdMS and incorporate information holding semantic on an entity.

The core identifier is intended to remain unchanged during the entities' lifecycle in the particular domain (namespace). Therefore, information categories (cf. Section 2.3.4) building a core identifier should provide non-volatile information, which excludes the context category as a possible approach. In contrast, the inheritance category offers high-security properties based on unchangeable entity information. However, the inheritance category requirements would obstruct the interoperability of the proposed IdM, since not all deployed Things have identity information (e.g., physically uncloneable function or watermarks) installed in their hardware.

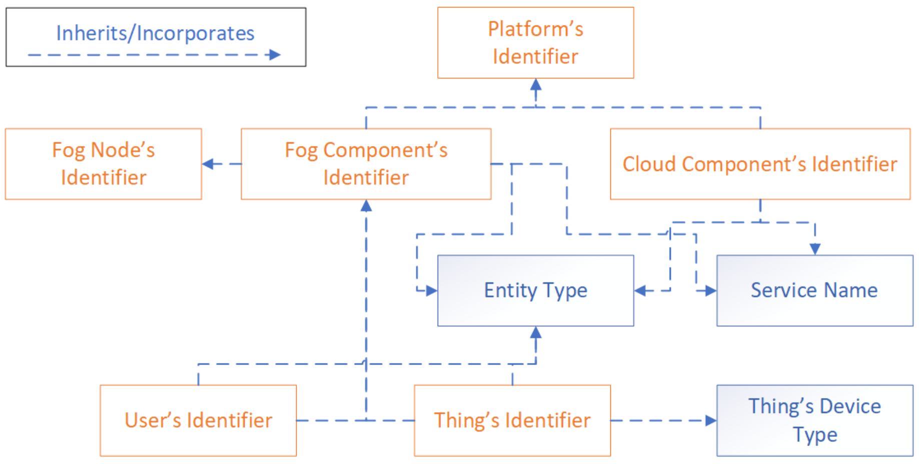


Figure 3.4: Core identifier association relations

The knowledge category (cf. Section 2.3.3) is incorporated in the designed solution through the generated identifiers by the CC and FC services and attached to the entity's core identifier. Therefore, the generated identifier has to be stored in the entity's memory or in the IoT system's trusted IdP (TNTA, FTA, or FTP). Generated identifiers are generated using a random function implemented in IdP, allowing the identifier's uniqueness and the low-level possibility for identifier collision within the IoT system. Besides the knowledge category, the association category provides additional information on the IoT system entity, describing the entity type, deployment position, and associations with other entity types (end-user, service) (cf. Figure 3.4). Associations allow efficient tracking of identities and their positioning within the IoT platform, since they contain the information on the domain in which the identity has been created.

Based on the described information categories utilization and identity naming scheme, core identifiers are:

1. `iot-platform-id.entity-type.service-name.`
   `cc-service-deployment-id (generated)` for CC service;
2. `iot-platform-id.entity-type.service-name.`
   `fog-node-id.fc-service-deployment-id (generated)` for FC service;
3. `iot-platform-id.entity-type.fog-node-id.`
   `faca-service-id.user-id (generated)` for User;
4. `iot-platform-id.entity-type.thing-device-type.`
   `fog-node-id.ftp-service-id.thing-id (generated)` for Thing.

The proposed naming scheme consists of two field types: enumerated (predefined, blue color) and generated (orange color). Generated fields are generated by IdPs and assigned to an entity, thus representing knowledge category information. Furthermore, these fields are used to describe associations between entities in TNDs. In the first association scenario, `fog-node-id` is embedded in all identifiers issued in the FC-based TND, uniquely identifying FN where IdP (FTA or FTP) is hosted. The second scenario involves using `fc-service-deployment-id` in user and Thing identifiers, documenting the identity of FACA and FTP services that represent their IdP, respectively. Remaining generated fields `user-id` and `thing-td` are used to uniquely address identity holders. However, these fields are not used for creating associations since user and Thing entities are not IdPs. Enumerated fields are used to provide more semantic on the given identity, describing the identity holder in more detail. These fields are:

- `iot-platform-id` documents the IoT platform's name;
- `entity-type` describes the type of the entity holding identity: CC service, FC service, User, or Thing;
- `service-name` is applied for CC and FC service identifiers, containing the deployed service's name (e.g., FTA, TNTA, or FTP);
- `thing-device-type` is used exclusively for Things' identifiers, documenting which device type the Thing is (e.g., Smart Outlet or Light Sensor), which is tightly coupled with access control procedures (cf. Section 3.3.2).

The interoperability of the proposed naming scheme allows the mobility of Things and users between FC deployments of the same IoT platform using the following fields: `fc-service-deployment-id`, `fog-node-id`, and `iot-platform-id`. Using `iot-platform-id` entity can be checked against affiliation to a particular platform. If different FC deployments belong to the same IoT platform, their identities can be easily derived from the `fc-service-deployment-id` and `fog-node-id` fields. Afterwards, the FC deployment's trustworthiness can be validated using procedures presented in Section 3.2.2.

3.2.2 *Trustworthy Fog-based IoT Networks*

Besides IdM provisioning, the establishment of TN incorporates mutual authentication between network entities, with the final goal of ensuring confidential communication between trustworthy entities (cf. Section 2.3.1). Mutual authentication relies on secure secrets (encryption keys, credentials) tightly coupled with identities in TND. For that purpose, ensuring secure dissemination and storage of secrets within TND through KMPs is essential. This section proposes a solution for the trust management in FC-based IoT environments. Thereby, the general approach for the trust relations management between identified entities in FC-based IoT networks, namely FC services, end-users, and Things is presented. While this section provides an overview for building the trustworthy FC infrastructure through authentication of FC services, due to the specific characteristics of the end-user and Thing entities, extensions regarding authentication for those entities are provided in the Sections 3.2.3 and 3.3.2, respectively.

Following the presented trust models in Section 2.3.2 and IdM deployment approaches in Section 2.3.3, an approach for ensuring trustworthy FC infrastructure (**TN 1**), enabling trust management for IoT network entities, is proposed. The proposed approach adheres to the TN requirements, namely **TN 4**, **TN 5**, and

TN 6, as well as the presented FC deployment model from Section 3.1.3. The proposed approach represents the backbone for trust management within the FC-based IoT system, while allowing an extension for ensuring secure communication for Things and end-users, presented in Sections 3.2.3 and 3.3.2.

The first step for defining the trust management solution is choosing a trust model. Due to the obligatory availability of CC-services, a centralized IdM deployment fails to satisfy the reliability (**TN 5**) requirement, since FC-based IoT networks have to ensure operability independently of CC services. Moreover, having a central entity managing all identities would represent a bottleneck for overall IoT service provisioning, negatively affecting scalability of the solution (**TN 6**). Therefore, the proposed solution is based on the distributed IdM and trust management through the FC services. However, FC services must establish and maintain trust relationships with the CC entities, whenever the CC services are available. To achieve that, the trust models mentioned in Section 2.3.2 have been analyzed. Direct trust models mostly involve SEKMP, providing efficient means for encryption keys dissemination, which is a significant benefit for the resource-constrained Things. However, a direct trust model introduces significant overhead for key storage and update procedures, resulting in cumbersome encryption keys management within the IoT system. The Web of Trust model relies on AEKMP, and the distribution of encryption keys and trust relationships is based on already established trust within the TND. Compared to the direct trust model, Web of Trust offers improved key distribution capabilities, since entities can rely on the distributed trust management in a TND for establishing trust. However, IoT networks' dynamicity (entities entering and leaving the TND) hinders the Web of Trust model's applicability in IoT, since it causes significant management effort to maintain the consistency of encryption keys in the Web of Trust network.

The hierarchical trust model involves the trust management distribution across multiple IoT network TNDs through the TAs. Moreover, one TND can be managed by one central TA, as proposed in Kerberos [Neu+05], or by multiple TAs, as in PKI [Cho+03]. One TA per TND approach proposed in Kerberos allows simpler identities and encryption keys management. Moreover, symmetric encryption-based key distribution in Kerberos introduces less performance overhead than asymmetric encryption involved in PKI digital certificates management. However, involving Kerberos would result in a mandatory requirement for Things to support symmetric encryption, which cannot be satisfied by all IoT devices [LC16]. In contrast, through digital certificates and digital signature-based authentication protocol, PKI represents a more distributed, scalable, and robust solution for trust management. The PKI's major drawback is the application of AEKMP, which is also hardly supported on IoT devices [KBL18a; Had+19]. However, PKI is a promising approach for the proposed solution since:

- It follows the hierarchical FC deployment model;
- It does not require the constant presence of all CAs to evaluate trust, enabling reliability (**TN 5**);
- It offers a standard procedure for handling digital identities (**TN 4**) through digital certificates;
- It does not introduce bottlenecks and SPoF to IdM through centralized IdP.

Due to those reasons, the hierarchical trust model and PKI are chosen as trust management and deployment models in our approach. The significant drawback of this design decision is the inability to deploy PKI on all Things due to their computational limitations. However, this drawback has been overcome through the introduction of security profiles, as described in Section 3.2.3.

The proposed trust management model is based on the divide-and-conquer strategy that separates IoT system in multiple TNDs (cf. Figure 3.5), categorized into two planes: (1) the **Cloud-to-Fog Trust Plane (CTFTP)** and (2) the **Fog-to-Thing Trust Plane (FTTTP)**. CTFTP is handled by **TNTA**, which represents the overall IoT system's root of trust, and maintains registry of all TNDs within the **CTFTP**, i.e. FTAs' digital certificates. FTTTP TNDs are handled by locally-deployed FC services, **FTA** and **FTP**. Hereby, roles and duties of the FTA and FTP are separated. **FTAs** represents the main CA for FTTTP TND, managing identities and digital certificates for the FC services and end-users within one Fog Trust Domain. **FTPs** is a TA in charge of managing the Things' trust. FTP's separation from FTA is mainly influenced by Things' incapability for supporting AEKMP and trust-chain validation through digital signatures. Therefore, FTP represents a first-hop computational capabilities provider for Things, enabling the security services relocation from Things to FNs capable of managing digital certificates and PKI. Features of FTP are described in detail in the Section 3.2.3.

For an entity to join the proposed trust management model in their pre-operational phase, authentication to the nearest CA, i.e. **TNTA**, **FTA**, or **FTP**, and issuance of the digital certificate is required. The issued digital certificate is afterwards used as identity authenticity proof throughout the system's lifecycle. Initial entity authentication (often referred to as trust bootstrapping) is implemented through the direct trust model. Depending on the trust plane, the trust bootstrapping procedure uses different approaches.

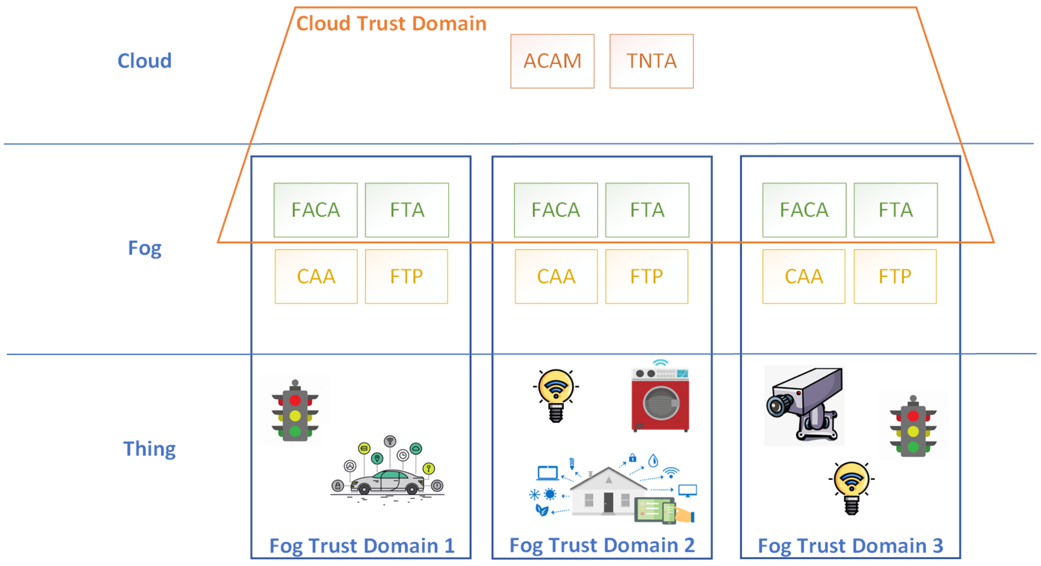


Figure 3.5: Trust Network Domains hierarchy

The CTFTP follows the procedure presented in Figure 3.6, with the primary goal to issue a digital certificate from TNTA to FTA. The initial handshake is based on offline-shared secrets: (1) TNTA's certificate, (2) application secret (AppSec) - key stored during FC service's image building, (3) instance secret (InSec) - key entered by the FN administrator during installation, unique to each FTA. The last secret that is not deployed with FN is the user secret (USec), which the FN administrator chooses during the FN installation. Besides being shared with FC service image, AppSec and InSec are entered in the TNTA database to whitelist the FTA instances that can be registered.

During its installation, FTA retrieves pre-shared secrets from its image and generates its identity (cf. Section 3.2.1), a public key pair, and a master key for initial trust bootstrapping using AppSec, InSec, and USec. In the next step, FTA registers itself by sending a request encrypted with TNTA's public key with its identity and hashed AppSec, InSec, and USec. Upon receiving the registration request, TNTA fetches stored FTA information and derives the same master key as FTA. Using this key, TNTA securely shares the generated Certificate Signing Request (CSR) token with FTA. Once having received the CSR token, the one-time secret for issuing a certificate, FTA creates a CSR request and decrypts the CSR token. The CSR request and plain-text CSR token are then sent to TNTA. By matching received and generated CSR token, TNTA can determine that FTA has used the same master key and issues a digital certificate upon validating the CSR.

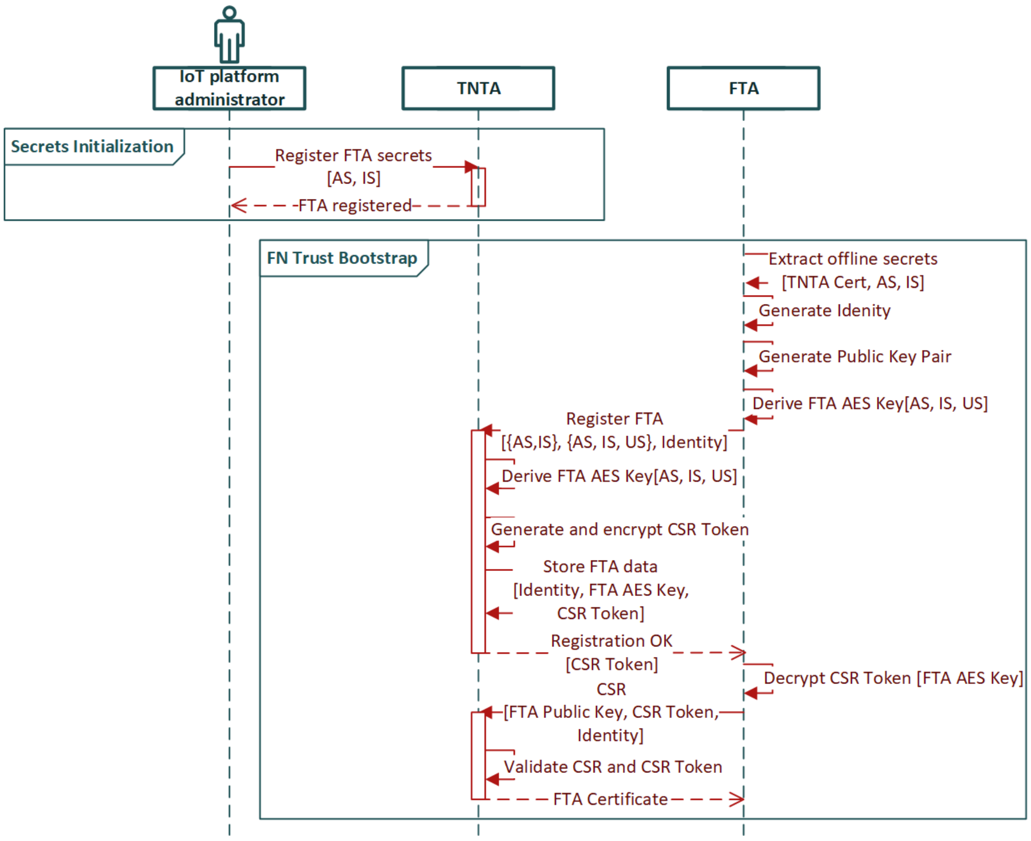

IoT platform administrator
TNTA
FTA
Secrets Initialization
Register FTA secrets [AS, IS]
FTA registered
FN Trust Bootstrap
Extract offline secrets [TNTA Cert, AS, IS]
Generate Idenity
Generate Public Key Pair
Derive FTA AES Key[AS, IS, US]
Register FTA [{AS,IS}, {AS, IS, US}, Identity]
Derive FTA AES Key[AS, IS, US]
Generate and encrypt CSR Token
Store FTA data [Identity, FTA AES Key, CSR Token]
Registration OK [CSR Token]
Decrypt CSR Token [FTA AES Key]
CSR [FTA Public Key, CSR Token, Identity]
Validate CSR and CSR Token
FTA Certificate

Figure 3.6: FTA trust bootstrapping

In the FTTTP, it is considered that FC components communicate over a local network. The FTA issues a digital certificate to other FC services (e.g., FACA or FTP) in this plane, as presented in Figure 3.7[20]. The pre-shared secret required for trust bootstrapping in FTTTP is the TNTA's certificate, allowing FTA's certificate validation. After generating its public key pair and identity, FACA retrieves FTA's certificate and tries validating it. Firstly, FACA checks if FTA's certificate is signed with TNTA's private key using TNTA's certificate. In the second step, FACA checks if TNTA is available and validates FTA's certificate at TNTA for revocation. If TNTA is not available, for example due to insufficient network bandwidth, FACA will trust FTA's certificate as is and retry validation at TNTA once TNTA becomes available. Once FACA has validated the FTA's certificate, it sends CSR to FTA and receives its digital certificate.

20 For demonstration purpose FACA is used as FC service. Still, the presented message exchange applies for all deployed FC services.

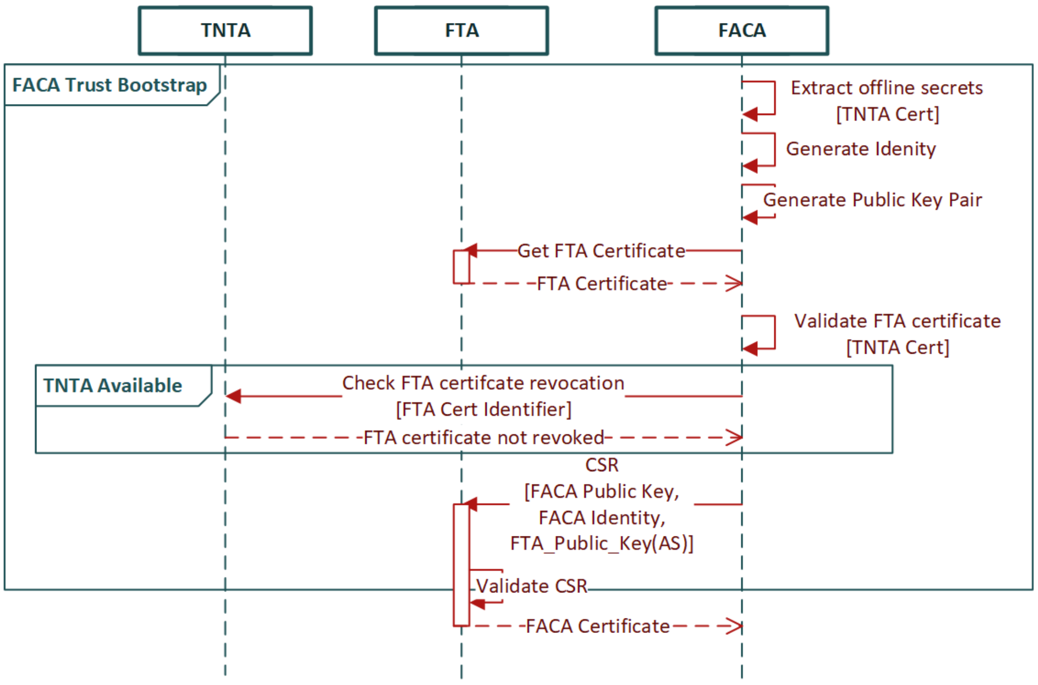

TNTA
FTA
FACA
FACA Trust Bootstrap
Extract offline secrets [TNTA Cert]
Generate Idenity
Generate Public Key Pair
Get FTA Certificate
FTA Certificate
Validate FTA certificate [TNTA Cert]
TNTA Available
Check FTA certifcate revocation [FTA Cert Identifier]
FTA certificate not revoked
CSR [FACA Public Key, FACA Identity, FTA_Public_Key(AS)]
Validate CSR
FACA Certificate

Figure 3.7: FC service trust bootstrapping

The proposed procedures ensure trust establishment for FC services. Trust bootstrapping for the remaining entities in IoT systems (users and Things) are covered in Sections 3.3.1 and 3.2.3, respectively.

Once having joined the PKI scheme and provided with digital certificates, entities step into the operational phase to communicate with each other in the IoT system. In this phase, FC services generate *Tickets* to mutually authenticate each other. The Ticket represents a unfalsifiable identity proof, containing information on communicating FC or CC service (Ticket issuer), invoked functionality, and message hash, signed with the issuer's private key. Upon receiving the Ticket, the communicating party can authenticate the message sender using PKI. In the FTTTP, each FC service creates Tickets for itself since the local FTA manages all certificates.

Since FTA is a TND gateway, meaning that only FTA registers its certificate at TNTA, FTA must create a Ticket for other FC services once they want to communicate to TNTA or ACAM. The procedure for trustworthy communication in CTFTP is presented and explained in Section 3.3.2 for the message exchange between ACAM and FACA.

Furthermore, the critical aspect of trust management is distributing certificate revocation information, represented through two standards: Certificate Revocation List (CRL) [Coo+08b], and Online Certificate Status Protocol (OCSP) [San+13]. CRL enables retrieval of revoked certificates from the CA, allowing the entity that verifies the certificate to check if the digital certificate is still valid. Compared to that, OCSP offers an API hosted by the CA that enables querying the certificate status. Due to that, information handled by a certificate verifying entity is smaller with OCSP than with CRL, which makes it more suitable for devices with limited memory, hence is a better option for resource-constrained Things.

However, OCSP implies constant availability of the CA, which is contradictory to the FC autonomy requirement (FC 5). Thus, the proposed certificate revocation notification strategy is split into two trust planes: CTFTP and FTTTP. CTFTP utilizes CRLs since they shift the revocation certificates management on TNTA. Thereby, FTA collects certificate revocation information from TNTA periodically or when TNTA becomes available. FTTTP aims to reduce computational effort, hence reducing hardware requirements for Things to join the proposed PKI solution. For that reason, OCSP API hosted in FTA and FTP services enables certificate status retrieval by entities in FTTTP, which excludes the need for managing certificate revocation information in other FTTTP entities.

The proposed IdM and PKI scheme enables the digital signature-based authentication procedures for various FC services, Things, and users by abstracting their identity through a digital certificate. The proposed scheme is based on the asymmetric encryption, which introduces performance load and additional computing requirements on the IoT system entities. This makes the scheme partially inapplicable to Things since it requires computational capabilities to support asymmetric encryption. This drawback is further discussed in Section 3.2.3, leading to the solution for further distribution of IdM using containerized FC services that can be deployed on various devices. This improves the distribution and scalability of the proposed IdM and allows the integration of resource-constrained Things into the proposed IdM scheme.

3.2.3 *Fog to Things Trust*

As described in Section 2.3.4, securing Things introduces several challenges to adapting traditional protocols and algorithms, primarily due to the Things' resource constraints. This creates motivation for establishing lightweight cryptography methods. However, security configuration management leads to an increase in Things complexity, which is not an acceptable approach for Things with strict size requirements and energy consumption. An alternative method for securing Things is to offload security operations from Things to computationally rich, edge layer devices residing in the same TND. Offloading simplifies securing configuration management for Things and minimizes the required energy consumption and computational capabilities to ensure TN.

Extending the proposed trust management solution (cf. Section 3.2.2) and integrating it with Things requires a computational requirements reduction. This primarily implies selecting better performing security protocols and algorithms, reducing encryption-caused computations while maintaining the same security levels. However, due to the Things' diversity, a one-size-fits-all solution is hardly achievable. For that reason, the application of multiple approaches is required to comply with Things' computational constraints for building trustworthy IoT services. The proposed approach for integrating Things into TND is twofold:

1. Evaluating various PKI schemes through IoT environment simulations to find the best performing one for the particular IoT application domain;
2. Offering encryption key exchange extensions in FTP for Things to integrate with, based on ”best effort” security using their available resources.

Firstly, PKI schemes efficiency and scalability are primarily affected by the applied encryption algorithms. Since the X.509 standard recommends the RSA encryption algorithm but does not require it as mandatory [Sta16], the application of the more efficient ECC could reduce the computational impact on the IoT devices [Bia+10; Sha+18; GMS15]. The efficiency of the ECC in reflected in the length of the encryption key to support the same security level: ECC achieves the 128-bits security level with a 256-bits key length, whereas RSA requires a 3072-bit long key for the same security level [BMZ13]. Moreover, proper optimization of the public-key and digital certificate management can improve the network transmission latency, requiring to send reduced MAC-layer messages during the certificate validation and key exchange procedures [Ted+20]. For example, RSA-based certificate validation requires the certificate owner to send the X.509

certificate containing the owner's RSA public key, signed by a CA [Bia+10]. Alternatively, the certificate management proposed in the Qu-Vanstone (QV) scheme employs implicit certificates, that hold the identity and the public key of a certificate owner [Cam22]. Therefore, the size of the certificate transmitted by the certificate owner is significantly reduced, minimizing the network message size and transmission latency [HNZ16].

The evaluation of the encryption scheme and certificate management impact on the PKI's efficiency and scalability motivated the design and development of the PKI simulator (cf. Section 4.5). The simulation is based on the network package simulator, emulating message exchanges between network entities, in this case, FNs and Things. Points of interest covered through the simulation are the performance aspects of the IdM and KMP schemes, i.e. CPU consumption, memory, and storage requirements, as well as the processing and network latency, required for the exchange of digital certificates and encryption keys along with their application for establishing a secured communication link. The simulation goal is to compare and analyze the benefits and drawbacks of the asymmetric encryption scheme: RSA and ECC, along with the certificate and KMPs: Diffie-Hellman (DH) and QV. The resulting set of the simulation scenarios includes (1) RSA with DH, (2) ECC with DH, and (3) ECC with QV. Besides the encryption algorithm and certificate management, another aspect introducing the latency in the certificate management is the certificate revocation validation. Therefore, each simulation scenario is extended with a certificate revocation strategy: CRL [Coo+08b] or OCSP [San+13].

Secondly, the above mentioned approaches for offloading security mechanisms from resource-constrained Things that cannot support the proposed PKI scheme to FNs, lead to the design of interoperable (**TN 4**) scheme for E2E security in IoT systems (**TN 3**). The proposed scheme relies on the deployed computationally-rich FN devices and offloading asymmetric encryption-required operations required by the PKI-based trust and IdM scheme proposed in Section 3.2.2. Offloading PKI operations is implemented through **FTP** services, which are deployed close to the Things. This ensures PKI operations offloading in a local network context, not as a remote CC service. Thus, FTP represents a bridge between PKI-based trust management between CC services, FC services, and Things that cannot support asymmetric encryption.

FTP manages the Thing's identity during its lifecycle, and therefore represents the Things in the IoT system. For that purpose, FTP establishes direct trust relationships with a Thing based on the Thing's computational capability and the security levels required by the IoT system. Initial trust establishment between Things and FTP is presented in Figure 3.8. The Thing first initializes its identity at FTP by sending its identity stored during the manufacturing process and supporting security profiles. Since this identity does not necessarily comply with the proposed identity naming scheme presented in Section 3.2.1, FTP generates identity for the Thing that will be used as Thing's core identifier in TND and sends it back to the Thing. Afterwards, Thing and FTP perform proprietary key exchange procedures based on the initialized security profile. Finally, FTP stores the exchanged key, the Thing's identities, and the generated public key pair to its keystore for the future use.

Once the trust between Thing and FTP is established, FTP handles the PKI-related cryptographic operations for the Thing, achieving mutual authentication between the Thing and other IoT network entities. These operations include validating other entities' digital certificates and digitally signing outgoing Thing's messages (e.g., sensor measurements or control messages).

Security profiles are determined based on similar approaches presented in [22b; Fan+17; GMS15; Bor+18], offering different data protection levels concerning data confidentiality and integrity. Security profiles (cf. Table 3.3) directly correspond to the trust levels in the IoT network, therefore indicating the trust reputation of Things and the transmitted data.

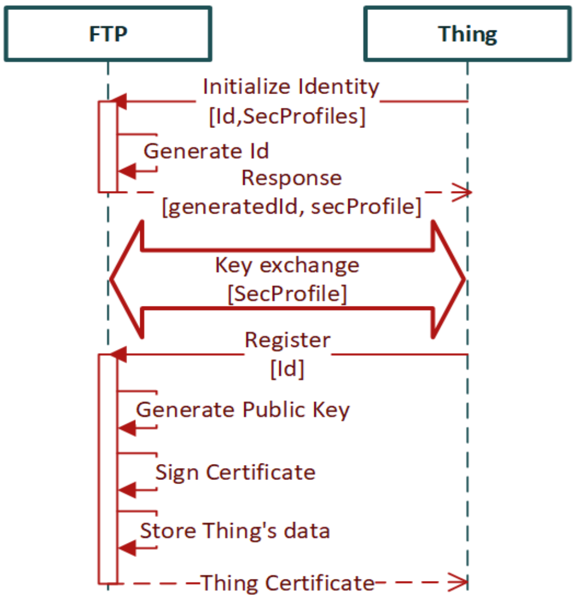


Figure 3.8: Thing trust bootstrapping

Table 3.3: Things Security Profiles[21]

Profile Name	Confidentiality	Integrity	Authenticity
No Security	No	No	No
Integrity	No	Yes	No
Symmetric, Insecure Keys	Yes*	Yes	Possible
Symmetric, Secure Keys	Yes	Yes	Possible
Asymmetric, Insecure Raw Public Key	Yes*	Yes	No
Asymmetric, Secure Raw Public Key	Yes	Yes	Yes

The **No Security** security profile does not support any of the CIA security principles, since the Things and FNs exchange plain text messages. Therefore, they do not establish any trust relationship utilizing encryption or hashing algorithms. The **Integrity** security profile requires a Thing to calculate message hash and attach it to the original package to allow FC service to validate data integrity. Still, there is no guarantee that the complete message has not been intercepted and retransmitted by another network node, due to the missing message authentication.

The **Symmetric, Insecure Keys** and the **Symmetric, Secure Keys** security profiles employ symmetric encryption as trust guarantee between Things and FNs, as incorporated in standard protocols IEEE 802.15.4 [Mon+07] and ZigBee [Fan+17]. By default, SEKMPs ensures data integrity and confidentiality, while authenticity can be embedded by attaching encrypted information on data sender and original information. However, the initial encryption keys distribution represents the critical factor concerning the support of CIA principles. Therefore, confidentiality provisioning in the **Symmetric, Insecure Keys** is supported only in cases when the initial key exchange has not been exposed to any attack, as for example in previous Zigbee protocol versions [Fan+17]. Compared to that, the **Symmetric, Secure Keys** security profile requires the application of protocols and key exchange algorithms that ensure secure key distribution. These protocols

21 CIA principles marked with the ”*” character in security profile indicate that the CIA principle is guaranteed only if the initial key exchange for that security profile has not been successfully attacked by a malicious network entity.

can be based on the principles described in Section 2.3.4, as well as approaches utilizing Out-of-Band channels like sound, visual, network signals on different frequency channels [COO13; 22j; LSA20].

The **Asymmetric, Insecure Raw Public Key** and the **Asymmetric, Secure Raw Public Key** security profiles employ data encryption using a public key and digitally signing data using a private key to ensure data confidentiality, integrity, and authenticity. These profiles follow the established TLS handshake-based protocols and solutions adapted to the Things' computational capabilities, as proposed by [Kot+13] and [RM12]. However, initial public key distribution between Things and FNs is a critical factor for the encryption scheme's trustworthiness. For that, the **Asymmetric, Insecure Raw Public Key** categorizes public key distribution strategies that are prone to the Man-in-the-Middle attack, leading to the untrustworthy data and entity authenticity.

3.3 Distributed Fog-based IoT Access Control

AC is the critical system security mechanism that prevents the unauthorized access to information and services, thus ensuring data confidentiality. AC incorporates AAA as building blocks for trustworthy identity verification, validation of granted access privileges, and monitoring access requests in the system, respectively. As documented in Section 2.4, AC mechanisms are well-researched and widely applied in CC and mainframe computer systems. Nonetheless, to comply with the distributed IoT environments, existing AC mechanisms need to be extended and adapted to be deployed at the edge of IoT networks.

The primary goal of this dissertation is the adaptation of AC mechanisms to enable automated and adaptive security policies in the IoT environment. Security policy adaptation and automation involves the analysis of information sensed by Things, determining the IoT environment's context and adjusting the security policies accordingly. Through that, the IoT system is capable of preventing unauthorized access with the reduced human effort required to configure security policies. To achieve that, FC is used as the underlying computing paradigm, enabling additional computational and storage resources in the local IoT environment. These resources are used to provide a low latency IoT environment's state analysis and adaptation of security policies. Moreover, if it is deployed on FN, ACC offers on-site access protection for Things.

In the scope of this dissertation, FC-based distribution of AC occurs through two software components - **ACAM** and **FACA** (cf. Section 3.1.3). Still, the desired distribution involves fulfillment of several requirements with regard to the functionalities provided within ACAM and FACA. Analyzed requirements are separated in three categories: (1) FC-inherited requirements (cf. Table 3.1), (2) requirements for IoT AC systems (cf. Section 2.4.5), and (3) support for C-A security policies (cf. Section 3.4). Despite separation through categories, the requirements overlap to a certain degree. An analysis on IoT AC requirements, as well as their relation to FC-inherited requirements of importance for the design of ACAM and FACA, namely **FC 5**, **FC 6**, **FC 7**, **FC 8**, **FC 9**, **FC 10**, and **FC 11** and context-aware security policy requirement is presented in the Table 3.4.

The enlisted requirements set impacts authentication and authorization mechanisms for the successful AC application. Meeting the requirements involves making several design decisions, allowing the trustworthy deployment of the AC services in the FC-based IoT systems. The design decisions, as well as their implications on the overall solution, are presented in the Sections 3.3.1 and 3.3.2.

3.3.1 *Security Policy Management*

The main goal of the security policies is providing rules for protecting the access to sensitive and critical system resources. In order to achieve that, security policies are mapped to the access policies collection through the description of security attributes. In IoT systems, two primary information producer and consumer types are a user and a Thing. Therefore, security policies enforcement and mapping has to provide a comprehensive security attributes set for describing both IoT network entity types.

Building the security attributes set in the scope of this dissertation is directly influenced by AC requirements enlisted in Section 3.3, namely **AC 3**, **AC 4**, **AC 5**, and **AC 7**. Moreover, the AC deployment model described in Section 3.1.3 affects the access policies, hence also the security attributes management.

Fine-granular settings for the security policies definition (**AC 4**) allow the configuration of access protection for the minimal set of IoT services. Through that, overall AC in the IoT system can be better adapted to the end-user needs. For example, a smart outlet in the Smart Home can be turned on only by its

Table 3.4: Internet of Things Access Control requirements

Label	Title	Relation to FC and C-A	Design implications
AC 1	Trust	FC 10, FC 11	Establishment and management of mutual trust relationships between FACA and ACAM based on TNTA and FTA components (cf. Section 3.2).
AC 2	Data management and privacy	FC 8	Validation of access rights in the local network, not as a remote CC service, hence private information are not transmitted over Internet without the previous application of security policies.
AC 3	AC policies management	FC 8, FC 9	Deployment and management of security policies on FNs, based on the preconfigured rules defined in ACAM.
AC 4	Fine-granularity	None	Ability to describe security policies based on multitude of security attributes originating from multiple sources (user, Things, IoT system context).
AC 5	Heterogeneity	FC 7	Standard generic scheme for security policies management and application for variety of Things.
AC 6	Reliability	FC 5	AC services provisioning in IoT environment through FACA, independently on the ACAM availability.
AC 7	C-A support	FC 6	Provisioning of integration points for C-A information to affect security policies management.

owner, whereas other users can just read the current consumption of a smart outlet. Achieving fine-granularity led to the definition of the Policy Anchor (PA), represented by an IoT system's service property, which will be guarded using AC. Following the heterogeneous nature of the IoT (**AC 5**), individual Thing's functionality has been chosen as a PA, allowing FACA to be deployed in different IoT application domains. Examples for such functionalities are turning a smart outlet on and off and changing color on a smart bulb. This decision, however, imposes the drawback of significant management and storage demand for access policies due to binding access policies to the single Thing's functionality. In this case, provisioning users' access to N Things, where each Thing has M functionalities, would lead to managing and storing $N * M$ access policies. This drawback has been partially prevented by defining grouped access policies in the **FACA**. Grouped access policies can be defined around similar Things' properties. In the FACA, the grouping is possible around: (1) multiple functionalities of one Thing, as well as (2) one or multiple functionalities of all Things of one type (e.g., smart bulbs or humidity sensors).

Further provisioning of fine-granular, heterogeneous security policies involves the description of end-users through generic information. Since IoT application domains serve different purposes, accompanied by various IoT systems' business logics, finding a standard set of user information would be very cumbersome. Therefore, our solution proposes generic user attributes as security attributes for the access policy definition. Generic attributes are used as placeholders for real user information in the IoT system where FACA is deployed. Therefore, based on IoT system requirements, generic attributes can be resolved to the user's age, nationality or system role.

To satisfy **AC 3**, **AC 4**, and **AC 5**, a particular AC model has to be applied. The first selection criterion for the AC model is the security policies management policy, that dictates privileges on who or what in the IT system can manage access policies. As described in Section 2.4.2, **MAC** and **DAC** are two dominant approaches for delegating security policy management privileges. **MAC** enforces centralized management privileges, enabling one or a couple of persons, usually system administrators, to manage security policies for the whole IT system. Hence, a central security policy is enforced for all IT system users, leading to more controllable security policies. In contrast, **DAC** allows all system users to assign security attributes and manage security policies. While this approach offers less manageability and, therefore, a higher probability of security policies misconfiguration, it offers higher flexibility for system users to adjust access rights without involving system administrators in the process.

IoT systems are highly-decentralized systems, with services deployed through Things and local controlling units (e.g., Smart Home gateway or monitoring units) in a variety of networks. Moreover, IoT systems require fine-grained and self-configuring access control mechanisms that adjust according to the dynamicity of the IoT environment's context. For those reasons, enforcing **MAC** in IoT systems leads to cumbersome and effort-demanding security policies, which would endanger the dynamic and availability of IoT services. Compared to that, **DAC** allows distributed security policies management, allowing the persons and network entities (C-A agents) to manage access rights. Although this approach can lead to a less secure IoT environment, its flexibility and supported dynamicity are prevailing factors to comply with the distributed AC implemented through FACA in the scope of this dissertation (**AC 3**).

The second selection criterion for choosing AC model is affected by the expressiveness of access policies, i.e. which information can be used as security attributes during the access rights validation. AC models **RBAC** and **IBAC** rely on subject attributes (e.g., role or identity) as an input for access rights description. Since IoT systems offer context information that can be used for access rights validation, thus requiring Context-Aware Access Control (C-A AC) (**AC 7**), **RBAC** and **IBAC** do not fit the enlisted requirements. **LATBAC** allows better description of security policies through mapping of security attributes to security levels. However, **LATBAC** is generally considered as **MAC**-based AC model, therefore not compatible with the previous choice for employing **DAC** in this solution.

ABAC supports security policy definition through generic attributes, which do not have to relate to subject information. Hence, ABAC is compatible with IoT context information as part of security policies. Supporting multiple generic attributes for a security policy definition represents a two-edged sword. Various information can be described through generic attributes and bound using logical clauses into fine-grained access policies (**AC 4**). However, this can lead to complex access policies, introducing higher effort for security policy management. Moreover, complexity can negatively affect the possibility of distributing access rights across the IoT system, since all access rights validation entities have to be able to support a complex validation logic. **CAPBAC** provides a somewhat different approach than previously mentioned AC models. As it relies on capabilities as the input for access rights validation. The capabilities are pre-calculated and pre-assigned to users based on security attributes. Therefore, users are required to provide proof of capability ownership to access the particular service. These characteristics allow more lightweight, easier-to-distribute application of **CAPBAC** in IoT systems, since a central entity assigns capabilities and that are validated directly at service providers, namely Things. While both **ABAC** and **CAPBAC** satisfy **AC 3** and **AC 4**, the capability pre-calculation requirement of **CAPBAC** represents a limiting factor for **AC 7**, as also stated by authors in [Oua+17]. Namely, context information impacted security policy adjustments can often occur, leading to an increased computing effort by AC modules. Moreover, capabilities re-calculation can often be useless since security policy adjustments can occur in periods when end-users do not use the IoT system. Therefore, while offering lower scalability and distribution capabilities, **ABAC** is chosen as AC model for this solution, satisfying **AC 3**, **AC 4**, and **AC 7**.

The proposed AC modeling approach supports the heterogeneity of IoT environments through the description of security policies using generic attributes, as well as applying AC policies on generic IoT services as PAs. However, provisioning interoperability between heterogeneous IoT environments involves the application of standardized language for access rights definition (**AC 5**). Moreover, sought language has to comply with previously chosen AC models, namely **DAC** and **ABAC**. XACML [oas22] enables standardized procedures for defining and validating fine-grained access policies, formatted in XML. AC policies management in XACML is decomposed into four modules: (1) Policy Information Point (PIP) offers services required for management of generic attributes, functionalities supported by IoT devices, as well as handling of user sessions, (2) Policy Administration Point (PAP) enables the definition and management of access policies, (3) Policy Decision Point (PDP) evaluates them in order to grant or deny access to IoT devices, and (4) Policy Enforcement Point (PEP) defines an interface required for the authorization of access requests and propagates them further to PDP. Through the integration of XACML into proposed solution interoperability between different IoT platform is ensured, since AC policy management is provided through the standardized message format between the standardized XACML modules, with the clear separation of duties.

3.3.2 *Access Control Distribution*

Having AC mechanisms deployed through **FACA** in separate networks requires establishing trust relationships within the local IoT network, that is TND, as well as within the overall IoT system, involving central AC services deployed in the Cloud through **ACAM**. Moreover, AC services must provide mechanisms for the end-users to authenticate in the IoT system. Once successfully authenticated, the user is considered as trustworthy by the IoT system and is issued with a session token holding security attributes. By presenting these security attributes, the user can gain access to IoT services through the access rights validation described in Section 3.3.1.

The TN and IdM deployment model described in Section 3.2.2 allows distribution of AC services to locally deployed FNs, enabling support for the requirements listed in Table 3.4. Namely, **AC 6** requires the availability of AC services despite the unreachability of CC services. For that reason, invoking ACAM services while initiating user sessions is not possible in every scenario, leading to endangered reliability of AC services in local networks (**AC 6**). In order to ensure the standalone operation of FACA, end-user IdM is deployed within the FACA. Through that, private user data is held exclusively in the local IoT network, that is on FN, enabling improved control over data and privacy settings (**AC 2**). At the same time, ACAM is used as the registry of the deployed FACAs and as a central entity for maintaining consistency between the security policies configured in FACAs.

Even though designed AC services are fully deployed to FN as FACA, they still depend on CC services: trust management in TNTA and security policies configuration in ACAM. In order to ensure trustworthiness of FACA services (**AC 1**), FACA needs to join IdM and PKI scheme described in Section 3.2.2 in the FTTTP (cf. Figure 3.5) by bootstrapping trust with the FTA component (cf. Figure 3.7). Once FACA has established trust with FTA, it enters the operational phase to retrieve security policies configuration from ACAM.

In its operational phase, ACAM and FACA maintain the configuration consistency of the security policies . This involves monitoring and management of the offered IoT services during the IoT system's runtime. Since access to the IoT services is controlled through **FACA's** authorization logic, the management of authorized Things' functionalities in FACA as part of the security policies is possible.

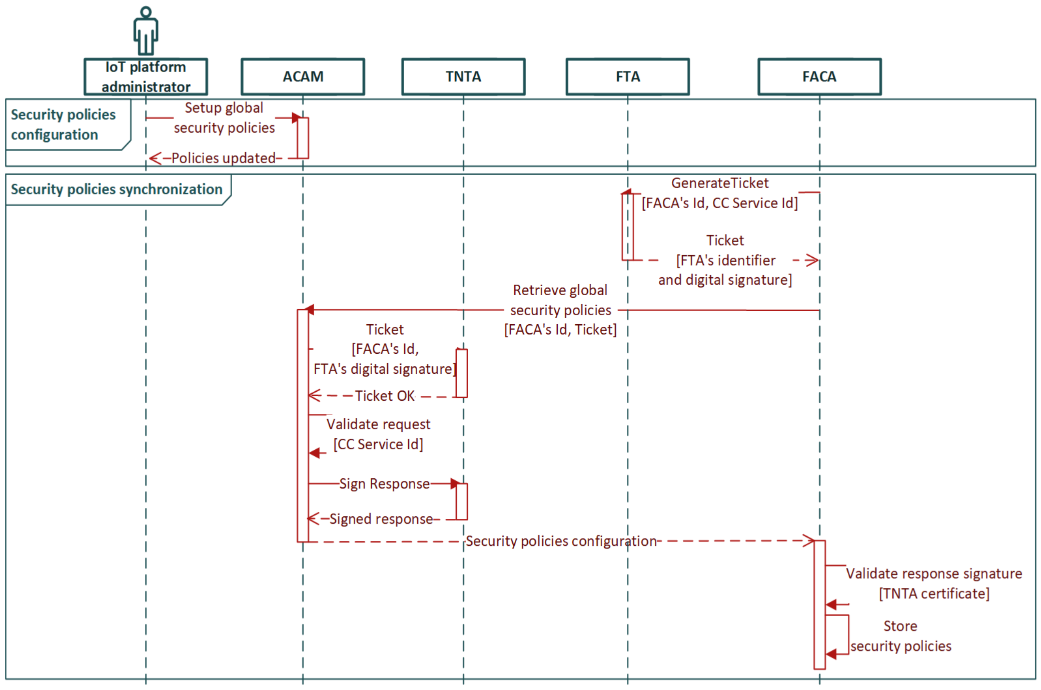

IoT platform administrator
ACAM
TNTA
FTA
FACA
Security policies configuration
Setup global security policies
Policies updated
Security policies synchronization
GenerateTicket [FACA's Id, CC Service Id]
Ticket [FTA's identifier and digital signature]
Retrieve global security policies [FACA's Id, Ticket]
Ticket [FACA's Id, FTA's digital signature]
Ticket OK
Validate request [CC Service Id]
Sign Response
Signed response
Security policies configuration
Validate response signature [TNTA certificate]
Store security policies

Figure 3.9: Global security policies synchronisation

The proposed solution bases maintaining security policies consistency by involving ACAM into security policies management through the supported Things' functionalities management, bringing additional benefits to the AC services deployment. The main motivation behind the CC-supported security policy management is to reduce the security risks on the IoT system in case of malfunctioning Thing functionality, i.e. detected security flaws or service malfunctioning. The establishment of a central control point for security policies on each FN enables IoT system administrators to react quickly and disable the targeted IoT services through ACAM, reducing the damage in case of a security breach. Therefore, FACA security policies must be derived from the ACAM's security policies, minimizing the local security policies divergence on FNs and allowing interoperable and consistent security policies between the deployed FACAs (**AC 5**).

To retrieve security policies from ACAM, FACA performs periodic long polling. Since long polling involves communication between services CFTFP and FTTTP, it represents cross-trust plane communication. Thus, FACA and ACAM have to mutually authenticate themselves, relying on the PKI described in 3.2.2 using *Tickets* for accessing each other services. As presented in Figure 3.9, before contacting ACAM, FACA requests a Ticket from FTA, since FTA is the TND gateway in FTTTP and through that recognized by TNTA in the Cloud. Once requested, FTA issues a Ticket signed with its private key to the FACA, enabling iy to access CC services. FACA sends a request and the Ticket to the ACAM, which validates the Ticket's integrity and authenticity in TNTA. If the Ticket is valid, ACAM generates a response and attaches a new Ticket with the TNTA's digital signature. Once FACA has received the response, it verifies the Ticket's digital signature using TNTA's certificate. If the digital signature is valid, FACA proceeds and stores the security policies.

The proposed global security policy management in ACAM relies on having Things' functionality as a security PA, as described in Section 3.3.1. Namely, ACAM utilizes the whitelisting of Things' functionalities to enable or disable provisioning of IoT services through FACA. Once the global security policies are defined and deployed to FNs (cf. Figure 3.9) , they are extended by local FC based security policies and specialized for a particular local IoT environment. Still, local security policies can be overruled by global security policies defined in ACAM by adjusting access policies in FACA according to ACAM's whitelists. For example: ACAM's global security policy can be:

A smart bulb can be switched on.

Additionally, FACA extends the global security policy by attaching security attributes from the local IoT environment:

A smart bulb can be switched on by a user who is older than 18 and has an administrator role.

Based on this extension, ACAM is not aware of local security policies but still can enable or disable them by updating Things' functionalities whitelists and maintain consistency of the configured security policies in each FACA.

3.4 Context Information Integration in IoT Access Control

Context information used to establish C-ASs enables a multitude of opportunities for enhancement of IoT systems. These opportunities contribute to further improvements in IoT solutions' usability and automation, leading to increased IoT adoption (cf. Section 2.5.5). In this dissertation, context is used to improve security policy management in IoT environments by integrating context information into AC mechanisms. **C-A AC** enables the automation of the access policies management, reducing users' effort for maintaining security policies. The importance of C-A AC has also been recognized by Ouaddah et al. [Oua+17] as one of the key requirements for future IoT AC systems (**AC 7**).

Building C-A AC requires building architecture and models capable of integrating context information into the AC scheme presented in Section 3.3.1. Context information is used as attributes in the ABAC model, describing the IoT environment's current state and impacting the access policies validation procedures. For that purpose, context information has to be collected, structured, and forwarded to FACA to enable context-aware configuration of security policies. To achieve that, the common C-AS design approaches presented in Sections 2.5.1 to 2.5.4 have been analysed. This involves the consideration of the requirements listed in Table 2.6, which are summarized into C-A AC design requirements in Table 3.5. These requirements, combined with the aforementioned design approaches, are used as guidelines for designing the aimed solution.

The initial step towards C-A AC design is to define the features and interaction options (cf. Section 2.5.1) that the C-A AC system will offer. Since C-A AC incorporates context information as part of security policies, which are configured by the end-user through the access rules, context enables a passive interaction option. This results in the AC system's capability to continually monitor the IoT environment and offer context information to the user to perform configuration actions on access policies, leading to the context-adaptive security policies.

Table 3.5: C-A AC design requirements

Label	Description
CA 1	Architecture layers and components
CA 2	Scalability and extensibility
CA 3	Application Programming Interface (API)
CA 4	Extended, rich, and comprehensive modeling
CA 5	Share information (real-time and historic)
CA 6	Automatic context lifecycle management

The establishment of C-A AC mechanisms involves defining the context model, which separates context acquisition and exposure to the application and users [Bet+10]. Building a comprehensive, rich context model (CA 4) that enables context services exposure through unique API (CA 3) requires identification of context information based on all entities producing context in the IoT environment (cf. Figure 2.4) - Things, places and end-users. For that purpose, the designed context model (cf. Section 3.4.2) is based on three C-A use cases, incorporating previously mentioned entities:

- Things' context is represented through their connectivity to the internet, which is handled by the **Connectivity Context-Awareness Agent (CCAA)**;
- **Behavior Context-Awareness Agent (BCAA)** collects the end-users context information, analyzing users' behavior and deducting behavior patterns over a longer period;
- Places are observed by the **Location Context-Awareness Agent (LCAA)** through users' location tracking and analyzing their positioning in the IoT environment.

These use cases also cover both context type dimensions (cf. Figure 2.4). The internal dimension is represented through BCAA, observing users' behavior, tasks, and interactions with the C-AS. The external dimension is observed by LCAA and CCAA, involving context measured by sensors, i.e. internet connection and user location.

The documented C-A AC design requirements (cf. Table 3.5) are analyzed in the upcoming sections. Satisfying the requirements involves the design decisions establishment with regard to C-A components architecture and integration (cf. Section 3.4.1), modeling context information (cf. Section 3.4.2), as well as the C-A components lifecycle (cf. Section 3.4.3).

3.4.1 *Context-Aware Components Architecture*

Context information collection, processing, distribution, and application in AC mechanisms require the design of C-A components in a generic manner, ensuring independence from the observed context information type. Moreover, C-A components have to satisfy requirements **CA 1** and **CA 2**, enabling a scalable, extensible, and layered components architecture, leading to the C-ASs that can easily integrate a variety of context information.

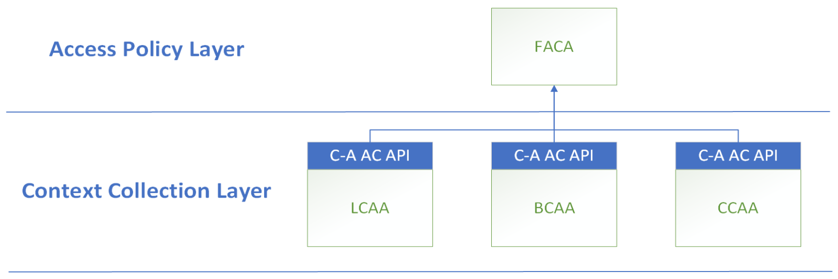


Figure 3.10: C-A layered architecture and components integration

Establishment of C-A AC mechanisms implies having the AC services deployed in FACA as the primary context information consumer. Thus, the separation of FACA from other C-A agents decouples AC services from the services for the collection of context information deployed in CCAA, BCAA, and LCAA, enabling a simpler distribution of C-A services. This leads to the layered architecture (**CA 1**), with (1) the context collection layer dealing with context collection and processing in C-A agents, and (2) the access policy layer enabling C-A AC based on the context information from the context collection layer (cf. Figure 3.10).

The separation of C-A components into two layers enables context information abstraction, so that FACA stays unaware of the context collection and processing details. This information remains encapsulated in the C-A agent which distributes the context information to FACA through the C-A AC API, which is common to all C-A agents. The C-A AC API (cf. Section 3.4.2) bridges the gap between low-level context information and access policies, enabling simpler integration of different C-A agent and accompanying context into AC mechanisms (**CA 3**).

As FACA consumes the information sent using the C-A AC API, it follows the **Context server** context acquisition approach (cf. Section 2.5.3) with explicit context model (cf. Section 2.5.1). These approaches enable straightforward C-A agents integration into AC mechanisms by adhering to the C-A AC API without any intervention on FACA's source code. This leverages the context information management complexity to C-A agents, guaranteeing FACA's capability to easily scale and extend to support various context sources (**CA 2**).

Since FACA enables FC-based AC mechanisms, C-A integration must not jeopardize the AC design requirements listed in Table 3.4 in any way. This affects the C-A agents deployments strategy since FACA has to operate independently of the CC services' availability (**AC 6**). For that reason, C-A agents are deployed along with FACA as FC services, ensuring minimal data transmission latency and reliability of AC-related features.

Furthermore, as C-A agents can collect and process private users' information (cf. Section 2.5.3), their deployment as FC services is beneficial for the overall system's security and privacy. As C-A agents process context information in a local network without its further propagation to CC services, users stay in control of their private information. Moreover, context information security has to be provided during the context information collection, modeling, and distribution [Per+14a]. To achieve that, C-A agents rely on the trustworthy environment through FTA and FTP. This guarantees the authenticity of the collected information since all connected sensors are provided with a digital certificate through FTP (cf. Section 3.2.3). Moreover, the authenticity of the context information distributed from C-A agents to FACA is enforced by digitally signing messages exchanged through the C-A AC API. This forces C-A agents to prove their identity with each transmitted message, eliminating possibilities to alter distributed context information and maliciously affect AC mechanisms.

Having C-A agents collecting and processing context information, it is essential to share this information with FACA (**CA 6**). Securing this information relies on the hierarchical trust through FTA components. To achieve that, FTA and FACA have to provide a support for automatic context management (**CA 7**), i.e. (1) registering and deregistering C-A agents, and (2) creation, update, and deletion of AC-related information through C-A AC API.

The lifecycle of each C-A agent is related to its registration and deregistration at FTA following the procedures described in Section 3.2.2. Once the C-A agent is registered at FTA , the C-A agent starts its operational phase (cf. Figure 3.11). In this phase, the C-A agent collects context information and distributes it to FACA through the C-A AC API. The C-A AC API's backbone is the context to attributes mapping, allowing FACA to consume all context information as attributes in the ABAC scheme. To achieve that, the C-A agent initially registers context attributes at FACA, thereby announcing which context information will be delivered in the future and can be used as part of access policies in FACA. Once registered, the C-A agent publishes context information through attributes value change notifications, which are stored in FACA's database for the future access policies validation. If the C-A agent detects any alteration of context information that FACA utilizes, the C-A agent sends a required for context attributes update, containing context attributes that should be created, deleted, or updated in FACA. Security of the message exchanges in the operational phase relies on the certificate established in the C-A agent's registration phase since all requests from C-A agent contain digital signatures that FACA and FTA validate for their authenticity.

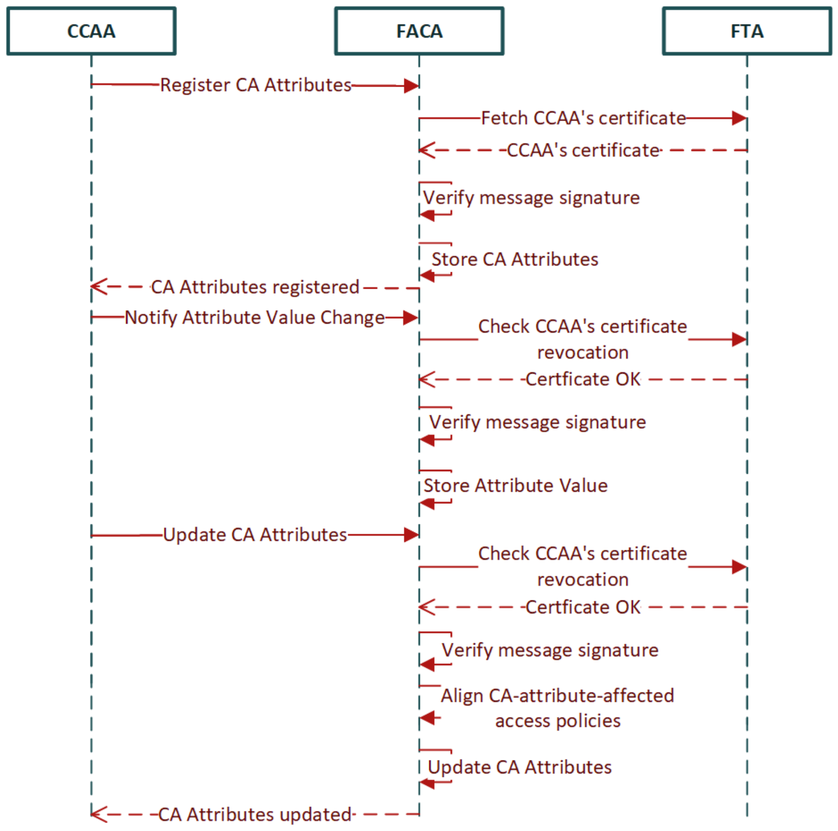


Figure 3.11: C-A agent's operational phase

C-A attributes-based access rights validation in FACA relies on the notified C-A attribute values. However, depending on the configuration and criticality of the operation that is being authorized in FACA, notified context information can be timely obsolete. For example, unlocking entrance doors to the Smart Home can be considered as highly critical operation, for which the most current C-A attribute value is required during authorization, whereas reading the room temperature is not that critical. To support security critical operations, C-A AC API enables registration of C-A agents interfaces, which can be contacted during the critical operations' authorization, as depicted in Figure 3.12. For FACA to contact C-A agent during the authorization, following requirements have to be fullfilled: (1) user has to define the access policy involving C-A attributes that are marked as critical and (2) C-A agents have to register interfaces for C-A attributes during C-A attributes registration. Once the requirements are satisfied, FACA invokes C-A agent's interface and performs the authorization using the latest C-A attribute value, which is not yet notified to FACA.

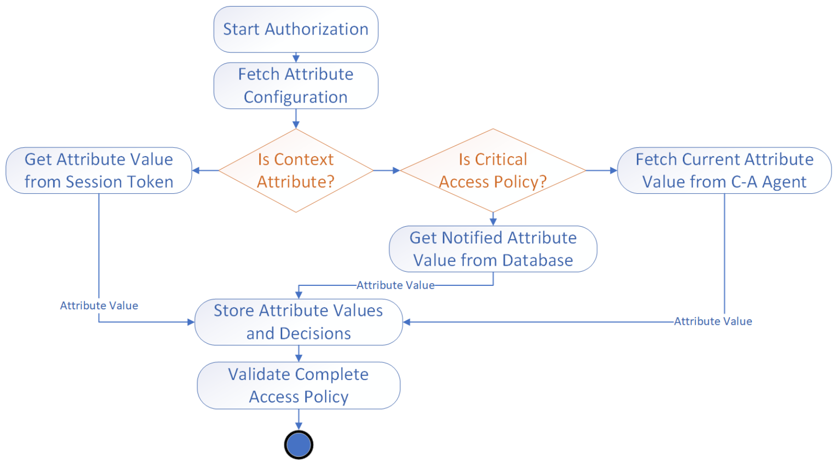

Start Authorization
Fetch Attribute Configuration
Is Context Attribute?
Is Critical Access Policy?
Get Attribute Value from Session Token
Fetch Current Attribute Value from C-A Agent
Get Notified Attribute Value from Database
Attribute Value
Attribute Value
Attribute Value
Store Attribute Values and Decisions
Validate Complete Access Policy

Figure 3.12: Authorization using C-A attributes

3.4.2 *Modeling Context Information*

Message exchanges presented in Section 3.4.1 heavily rely on the C-A AC API to support the C-A AC mechanisms' scalability and extensibility with various context information. Hence, modeling the C-A AC API plays a critical role for the designed solution, which should result in a comprehensive, rich, and extensible model (**CA 4**) exposed by FACA's API (**CA 3**).

To satisfy these requirements, modeling the API using an ontology-based approach is applied. The key factor for choosing ontologies amongst other modeling methods presented in Section 2.5.4 is mostly due to the existence of numerous tools and languages (e.g., OWL, RDF, and RDFS) that enable straightforward understanding of complex relations between context information [Wan+04].

Designing context ontologies that cover a wide range of context information is a challenging task [BDR07a]. Requirements for building context ontologies listed in publications [KM03; PRL09] are: simplicity, extensibility, generality, expressiveness. Furthermore, Perera et al. [Per+14a] emphasize the importance of defining the ontology scope and domain within a context model, as well as reusing existing ontologies.

Designing a simple and generic ontology that allows integrating various C-A agents into AC mechanisms implies splitting the C-A AC API ontology model into two smaller ones: FACA ontology and C-A agents ontology (cf. Figure 3.13). This separation enables a generic approach that allows FACA to easily extend its capabilities to support various C-A agents by abstracting context information through attributes in the ABAC scheme. Furthermore, C-A agents domain models do not have to be ontology-based, as long as they support the C-A agents ontology and can map the internal context model to the one implied by FACA.

Mapping various context information to attributes requires a comprehensive definition of the C-A agents ontology. Therefore, this ontology includes basic context attribute properties and context QoS properties (cf. Section 2.5.2), which can affect AC mechanisms. As documented in Figure 3.13, the C-A agents ontology is separated into two major parts: **Configuration** and **Value Notification**.

The Configuration ontology provides initial information on the context that will be delivered to FACA in the future and enable access rights validation. On the one hand, this information provides identifiers for future Value Notification messages: attribute name, context type. On the other hand, with the goal of enabling real-time context information during authorization procedures, each C-A agent can expose an interface which FACA can contact during the authorization to fetch the latest C-A attribute value. This interface's address is defined in the field *Attribute Evaluation Route*. Context QoS Configuration data define context information which FACA afterwards uses to register new attributes and allow a user to create access policies using context information:

1. A certainty (confidence) range described as maximum and minimum value;
2. Time constraints represented through distribution pattern, maximum time between two value notifications, and notified values validity;
3. Data type depending value constraints, i.e. string or numeric;
4. Type of the access rule that can be established based on provided context information.

Value Notification messages occur after completing the C-A attribute registration and configuration, thereby delivering context information that is used during C-A-based authorization. The notified values are mapped to the previous Configuration and AC policy information using the registered attribute name. The QoS aspect of context information is covered through the certainty field, enabling C-A Agent to express confidence in the notified value.

Support for C-A attributes through FACA ontology relies on ABAC and the XACML scheme described in Section 3.3.1. However, context information could not be directly mapped to the existing XACML capabilities since C-A attributes contain QoS aspects, which are not present at user-related attributes (role, age, nationality). Therefore, XACML has been extended with the **Context** access rule type. This access rule type consists of:

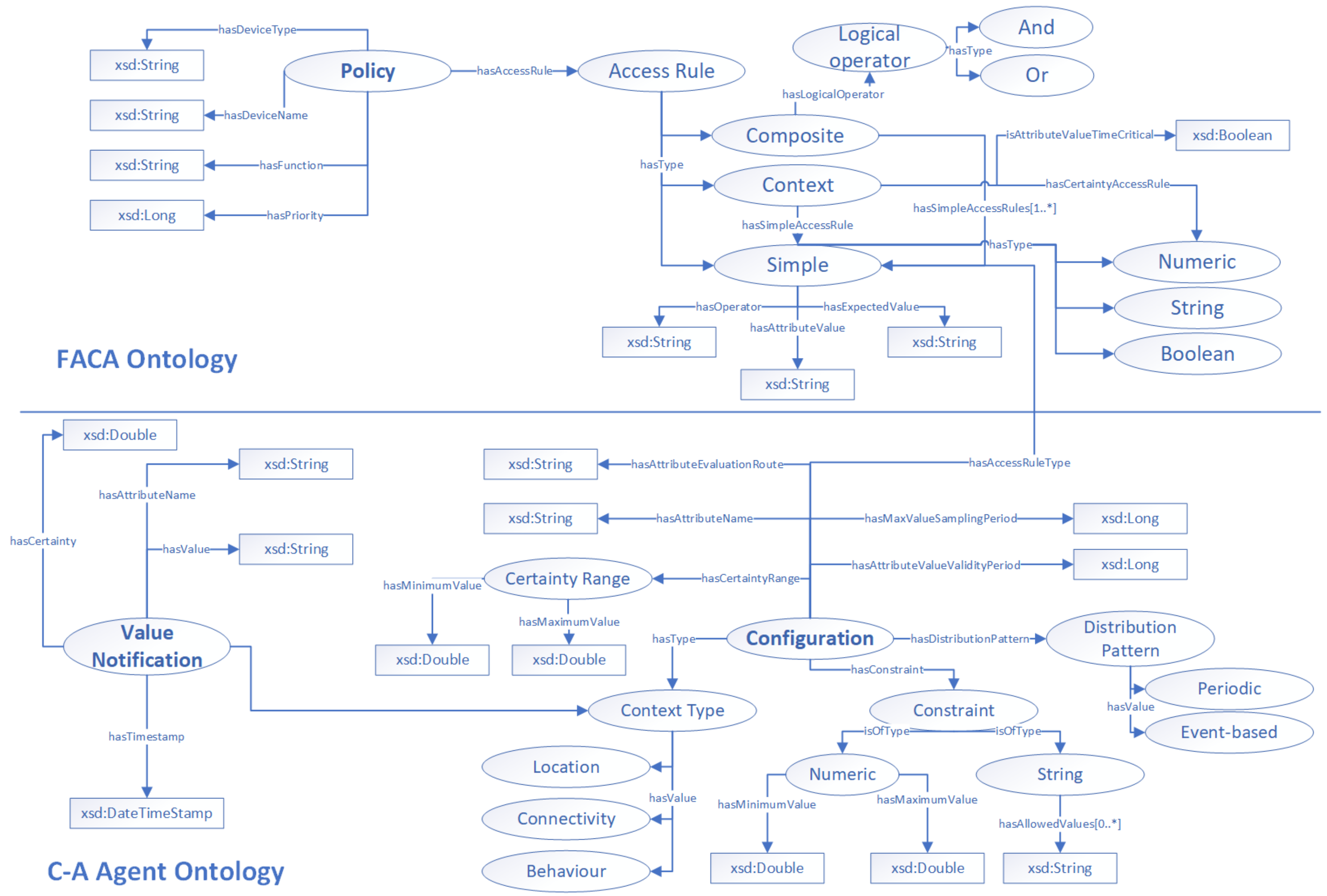


Figure 3.13: C-A AC API ontology

1. Numeric access rule that evaluates certainty field from Value Notification messages;
2. Simple access rule that evaluates C-A attribute values and is defined in the Configuration message;
3. Indication if the latest C-A attribute value needs to be fetched from C-A agent during the authorization.

3.4.3 *Context-Aware Components Lifecycle*

Provided description of the FACA's lifecycle (cf. Section 3.4.1) documents design decisions concerning the integration of the context information into AC mechanisms. Each C-A agent has its lifecycle, during which it collects information from sensors or middleware and shares the real-time and historical context information with FACA (**CA 5**) described in Section 2.5.4.

Table 3.6 presents the summary of the C-A components lifecycle design. Each C-A component is described in terms of the context acquisition, modeling, and reasoning aspects. The context information distribution aspect is the same for all C-A agents, following the push approach using C-A AC API (cf. Section 3.4.1). Furthermore, a brief description for each C-A agent's lifecycle is provided.

BCAA tracks user behavior patterns and analyses them to estimate users' expected actions in an IoT environment. The input for behavior patterns analysis includes the events describing (1) user controlling Things and (2) periodic measurements (e.g., temperature or humidity measurement). BCAA tries to find behavior patterns' correlations between users and their usage of IoT services based on these events. BCAA outputs the users' behavior patterns that are identified and its confidence for the patterns to repeat in the future, which FACA then uses for defining security policies.

Compared to CCAA and LCAA, BCAA has a slower learning curve since the events indicating user's behavior can occur quite randomly (e.g., every few days or weeks). For that reason, context information provided by BCAA is operationally categorized as ”profiled” one (cf. Section 2.5.2), meaning that they change over time with a low frequency. Through the events pushed from the middleware, BCAA acquires context information instantly and periodically. However, through the events' analysis, BCAA derives context information through unsupervised learning reasoning models, which is then published to FACA.

Table 3.6: C-A services lifecycle aspects

Component	Acquisition					Modeling	Reasoning
	Source	Responsibility	Frequency	Sensor Type	Process		
FACA	Context server	Push	Interval	None	Sense	Ontology	Rule-based
BCAA	Middle-ware	Push	Instant + interval	None	Derive	Ontology	Unsupervised learning + fuzzy logic
CCAA	Sensor	Pull	Interval	Virtual	Sense	Object-oriented	Probabilistic
LCAA	Middle-ware	Push	Instant	None	Derive	Object-oriented	Rule-based

CCAA monitors internet connection and availability to the Cloud servers by periodically (every 10 seconds) calling the exposed health-check route, exposed by the Cloud server. The connection to the Cloud Server can fail due to multiple reasons, such as network timeout, bandwidth congestion or Cloud Server overload, resulting in temporary or long-lasting connectivity problems. Therefore, CCAA considers the last ten connections to the Cloud to estimate the probability of a successful connection.

Since CCAA acquires the required context information by pulling data directly from a virtual sensor (the network interface), it is considered to be using the primary context. Moreover, dealing with the estimated certainty of connectivity from the last ten intervals, CCAA performs a probabilistic reasoning model. Finally, having a relatively simple context model, CCAA handles context information through an object-oriented approach, simplifying the context lifecycle management implementation in Java programming language.

LCAA tracks users' positioning within the IoT environment and describes their current location through a 4-tuple: building, floor, room, and zone. User position tracking is based on Bluetooth beacons distributed over an IoT environment and configured through the 4-tuple mentioned above. The Happy Bubbles[22] open-source project is used for the presence detection near a Bluetooth beacon, which determines the user's current position.

By receiving processed context information from the middleware (Happy Bubbles), LCAA instantly consumes secondary context information as soon as the user's presence is detected. Afterwards, LCAA applies rule-based reasoning to derive the user's location through pre-configured Bluetooth beacons positions. Finally, beacons configurations and user's presence information are modeled using an object-oriented approach, since context information mostly relies on 4-tuples and does not possess too many interrelationships, so that more complex context models have to be applied.

22 `https://www.happybubbles.tech/`, last access May 2, 2022

3.5 Summary and Findings

This chapter described design decisions for the developed trustworthy, FC-based, C-A AC solution and covers the main building blocks of the designed solution concerning their requirements and resolution approaches. First of all, the deployment strategies for FC services as well as their implications on the development of TN, AC, and C-A services have been evaluated in Section 3.1.2. The most critical implication is the requirement for the uninterrupted operability of the FC services' independently on Cloud Server availability, which hindered the application of many established TN and AC solutions.

Sections 3.2 and 3.3 contain the design decisions concerning TN and AC services, respectively. Based on the inherited requirements from FC and collected requirements from the current research in these areas, several protocols, frameworks, and models have been analyzed, leading to the definition of the novel protocols for TN and AC in FC-based IoT environments. Finally, C-A integration in AC mechanisms has been evaluated in Section 3.4. The variety of C-A factors that can impact AC in IoT dictated a comprehensive analysis of the approaches for context information management. The final result is defined through a generic, extensible, ontology-based data model that enables the integration of context information into ABAC-based AC mechanisms.

Conclusions on the design decision with regard to initial requirements of the main solution areas have been summarized and presented in the following Tables: (1) Table 3.7 for FC, (2) Table 3.8 for TN, (3) Table 3.9 for AC, and (4) Table 3.10 for C-A. Each design decision is presented with its label, brief solution description, and a reference to the Section where more details concerning the design decisions can be found.

Table 3.7: Fog Computing solution requirements and design decisions

Label	Title	Section	Solution Design
FC 1	Highly-virtualized deployment	Section 3.1.3	Software components' deployment using containers - lightweight service execution environment.
FC 2	Hierarchical organization	Section 3.1.3	Splitting software components into Cloud, Fog, and Thing hierarchy. Also, FC services deployment on different FNs, with different computational constraints, based on service complexity and dependencies.
FC 3	Low network latency	Section 3.1.3	Deployment of time-critical services close to the IoT network's edge, reducing data transmission distance.
FC 4	Scalability of FC network	Section 3.1.3, Section 3.2.2	Redundant components deployment using containers and establishment of lightweight TN concepts for achieving trust.
FC 5	CC services independence	Section 3.2.2, Section 3.3.2	As described in AC 6 and TN 5 requirements analysis.
FC 6	Location and Context-Awareness	Section 3.3.1, Section 3.4	As described in AC 7 and all CA requirements analysis.
FC 7	Interoperability	Section 3.2.2, Section 3.3.1	As described in AC 5 and TN 5 requirements analysis.
FC 8	Data security and privacy	Section 3.3.2, Section 3.3.1	As described in AC 2 and AC 3 requirements analysis.
FC 9	E2E security	Section 3.2.3, Section 3.3.1	As described in AC 3 and TN 3 requirements analysis.
FC 10	Trustworthy FC infrastructure	Section 3.2.2, Section 3.3.2	As described in AC 1 and TN 1 requirements analysis.
FC 11	Identity Management	Section 3.3.2, Section 3.2.1	As described in AC 1 and TN 2 requirements analysis.

Table 3.8: Trustworthy Networking requirements and design decisions

Label	Title	Section	Solution Design
TN 1	Trustworthy FC infrastructure	Section 3.2.2	Achieving trust between users, Things and services using digital certificates for mutual authentication and encryption keys exchange.
TN 2	IdM	Section 3.2.1	Designed common naming scheme for users, Things and services in IoT system.
TN 3	E2E security	Section 3.2.3	Establishment of best-effort trust levels, based on the direct trust model between Things and closest TN services in the network.
TN 4	Interoperability	Section 3.2.2	Application of standard solutions for managing digital certificated - PKI, X.509, and ECC, as well as common IdM scheme within the IoT platform (TN 2).
TN 5	Reliability	Section 3.2.2	Capability of managing trust within local network using PKI and FTA as the trust anchor when CC-services are unreachable.
TN 6	Scalability	Section 3.2.2	Deployment of additional, lightweight FTP component that handle Things digital certificates, enabling horizontal scalability for numerous interconnected Things.

Table 3.9: Access Control solution requirements and design decisions

Label	Title	Section	Solution Design
AC 1	Trust	Section 3.3.2	Establishing trust relationships between CC and FC services through digital certificates and centralized whitelisting of the allowed Things' functionalities through ACAM.
AC 2	Data management and privacy	Section 3.3.2	Enforcing security policies in the local network, disabling data transmission to the Cloud Servers and ensuring users' control of the data.
AC 3	AC policies management	Section 3.3.1, Section 3.3.2	Management of security policies in the local network using ABAC model, based on the configuration deployed from ACAM.
AC 4	Fine-granularity	Section 3.3.1	Application of ABAC model, allowing the generic description of users' characteristics. Moreover, using single Thing's functionality as the PA.
AC 5	Heterogeneity	Section 3.3.1, Section 3.3.2	Using generic placeholders for defining security policies - user and IoT environment attributes, as well as Things' functionalities as PAs.
AC 6	Reliability	Section 3.3.2	The ability of FACA to enforce security policies without contacting Cloud Server, operating completely standalone once configured.
AC 7	C-A support	Section 3.3.1	Application of ABAC model, enabling the integration of context information about the IoT environment as attributes in the security policies.

Table 3.10: Context-Awareness solution requirements and design decisions

Label	Title	Section	Solution Design
CA 1	Architecture layers and components	Section 3.4.1	Splitting C-A features between FACA (access policy layer) and C-A agents (context collection layer). C-A agents sense and forward context information, which FACA applies in security policies.
CA 2	Scalability and extensibility	Section 3.4.1	C-A agents' straightforward addition through C-A AC API allows extensibility and scalability through deployment on various FNs.
CA 3	Application Programming Interface (API)	Section 3.4.1, Section 3.4.2	Common C-A AC API that allows generic interfaces for context information integration through the attributes abstraction.
CA 4	Extended, rich, and comprehensive modeling	Section 3.4.2	Ontology-based context model that extends XACML to support generic C-A-based attributes in ABAC.
CA 5	Share information (real-time and historic)	Section 3.4.3, Section 3.4.1	Periodic or event-based context information publishing from C-A agents through C-A AC API to FACA.
CA 6	Automatic context lifecycle management	Section 3.4.1	C-A agents can integrate with FACA by managing (registering, revoking) their identity and digital certificate at FTA (setup, termination phase) and securely publishing context information to FACA in the operational phase.

4 Implementation

Following the design decisions presented throughout Chapter 3, provisioning TN and C-A AC involves multiple solution steps in the network security and C-AS design areas. Hence, Chapter 4 provides further insights into the implementation of the envisioned IoT framework - COSYLab. Through the COSYLab, the presented solution design is implemented and validated using a Smart Home IoT application domain. Nevertheless, the developed solution is applicable in other IoT application domains, involving various IoT devices from different hardware vendors.

The upcoming Sections 4.1 - 4.7 document the key aspects concerning implementation of COSYLab. Firstly, the implementation scenario overview is provided in Section 4.1, followed by the software architecture and communication protocols description in Section 4.2. The FC and CC services deployment strategy details are provided in Section 4.3. Afterward, TN provisioning implementation details are documented in Sections 4.4 and Section 4.5, with a focus on (1) developed TN services and protocols, and (2) PKI performance simulations. AC implementation information is presented in Section 4.6, followed by the extensions for integrating C-A factors into AC (cf. Section 4.7).

4.1 COSYLab Smart Home Scenario

In order to implement and evaluate the designed protocols and models described in Chapter 3, the COSYLab framework is implemented, representing the Smart Home IoT environment. The implementation scenario involves a Smart Home with multiple rooms spread across multiple floors. Each room is featured with several Things that are connected with the Smart Home gateway, which is hosted on a FN. A family consisting of two adult persons and several children are envisioned as Smart Home system users. They control the Smart Home appliances and fetch the sensor readings several times per day. Based on the Cisco Annual Internet Report[23], it is assumed that each person in a Smart Home actively uses around nine Things, resulting in the estimated number of 50 present Things in a Smart Home. The Smart Home is envisioned as an FC-powered IoT environment. For that reason, Smart Home operations have to be resilient to the unavailability of CC services. Still, to optimize and maintain a consistent state of TN and AC services, Smart Home FC services are synchronizing with CC services whenever the connection to CC services is available.

For the development of the software components: (1) CSWA, ACAM, and TNTA in CC environment and (2) FACA, FTA, FTP, LCAA, BCAA, and CCAA in an FC environment, the following key software frameworks, libraries, and message brokers are used:

- MongoDB[24] for information persistence in database,
- Spring Boot[25] framework for business logic, Advanced Message Queuing Protocol (AMQP) and Representational State Transfer (REST) interfaces,
- AngularJS[26] for presentation of information in browser, and
- RabbitMQ[27] for AMQP communication between FC services.

Throughout the implementation and evaluation of the COSYLab, the developed software components have been deployed on multiple devices with various computing and storage capabilities spreading through the Cloud - Fog - Things

23 https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report/air-highlights.html, last access May 2, 2022
24 https://www.mongodb.com/, last access May 2, 2022
25 https://spring.io/projects/spring-boot, last access May 2, 2022
26 https://angularjs.org/, last access May 2, 2022
27 https://www.rabbitmq.com/, last access May 2, 2022

continuum. The Cloud Server is hosted as a PaaS-based Linux Ubuntu 18.04[28] virtual machine with 2.8GHz dual-core CPU and 4GB RAM. FNs are represented through three different devices, where each FN device is installed with a Linux Ubuntu Server 21.04[29] operating system. Used FN devices are:

1. Raspberry Pi4 model B[30], with 1.5GHz quad-core CPU and 4GB RAM;
2. Raspberry Pi3 model B+[31], with 1.4GHz quad-core CPU and 1GB RAM;
3. Raspberry Pi Zero[32], with 1GHz single-core CPU and 512MB RAM.

Through the FC services deployment on different FNs, it is possible to evaluate the feasibility, risks, costs, and benefits of FC services deployment. For that reason, Raspberry Pi4 and Raspberry Pi3 represent FNs with more computational and storage capabilities. They are envisioned as FN that can host multiple FC services and provide better performances for time-critical operations. Raspberry Pi Zero represents resource-constrained FN, offering worse performances but still providing capabilities to distribute processing of FN services, minimizing processing bottlenecks and SPoF risks.

Things are represented through the NodeMCU-powered ESP32 development board[33], and provided with 448 KByte ROM, 520 KByte SRAM, and Xtensa 32-bit LX6 dual-core processor. ESP32 supports 2.4 GHz wireless protocols: IEEE 802.11 b/g/n/e/i and Bluetooth protocols, i.e., BR/EDR and BLE.

4.2 Software Architecture

In order to fulfill identified requirements and achieving the defined goals, several design decisions were taken as justified through Chapter 3, leading to the architecture presented here. In this section, a general overview of the components organization within the COSYLab, as well as communication protocols and interfaces between them, is documented.

28 `https://releases.ubuntu.com/18.04.5/`, last access May 2, 2022
29 `https://ubuntu.com/blog/ubuntu-server-21-04`, last access May 2, 2022
30 `https://www.raspberrypi.org/products/raspberry-pi-4-model-b/`, last access May 2, 2022
31 `https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/`, last access May 2, 2022
32 `https://www.raspberrypi.org/products/raspberry-pi-zero/`, last access May 2, 2022
33 `https://nodemcu.readthedocs.io/en/dev-esp32/`, last access May 2, 2022

The developed software components are based on the existing Smart Home framework developed at the University of Vienna, allowing IoT devices management, sensor data collection and provisioning of collected information (e.g., current temperature or air quality measurements) to the users. Components of the Smart Home framework are colored green in Figure 4.1. Integration of the Smart Home framework and the Things is provided through the Home Assistant[34] home automation framework. Using the Home Assistant, integration with several Things types is provided: temperature, light, and humidity sensor. Home Assistant operations are exposed through the API of the Fog Controller component, representing the central point for the IoT services provisioning. Therefore, the provisioned AC, TN, and C-A services are integrated with the Fog Controller to provide security mechanisms as Smart Home framework's extensions.

Figure 4.1 shows the COSYLab framework, extended with software components for security and C-A services provisioning. Developed software components depicted with orange and blue boxes in Figure 4.1 represent security modules deployed in Fog and Cloud environment, respectively. These components are integrated using the application layer protocols, namely HTTP and AMQP. Marked with blue color, HTTP-based communication is used for interactions between FC and CC components, between CC components, as well as between Fog Controller and FACA, due to the performance reasons (cf. Section 5.4.2). HTTP-based communication relies on REST enabling a standardized approach for the integration of communicating components. Orange-colored communication links represent AMQP-based communication between FC components, allowing loosely-coupled, synchronous, and asynchronous communication between FC components. Used message format in case of both HTTP and AMQP communication is JSON[35], offering a lightweight data-exchange.

To enable targeted TN, AC, and CA services in IoT environment, developed software components are grouped based on the provided service. Therefore, details on each developed component are presented in the upcoming sections:

- Section 4.4 provides implementation details on TN provisioning, accompanied by description for TNTA, FTA, and FTP components;
- Section 4.6 provides implementation details on AC provisioning, accompanied by description for ACAM and FACA components;
- Section 4.7 provides implementation details on C-A provisioning, accompanied by description for LCAA, BCAA, and CCAA components.

34 https://www.home-assistant.io/, last access May 2, 2022
35 https://www.json.org/json-en.html, last access May 2, 2022

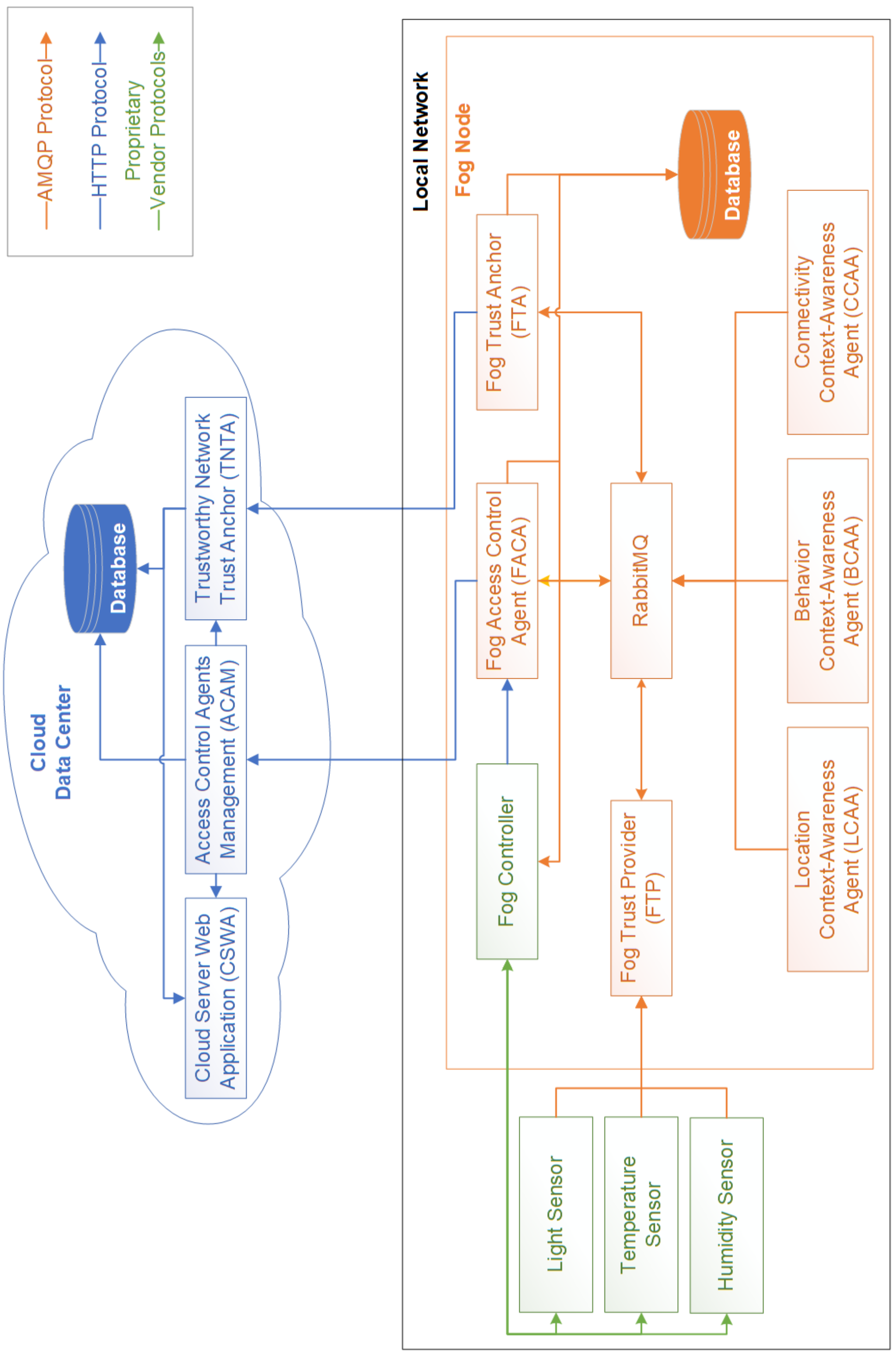

AMQP Protocol
HTTP Protocol
Proprietary Vendor Protocols
Cloud Data Center
Database
Trustworthy Network Trust Anchor (TNTA)
Access Control Agents Management (ACAM)
Cloud Server Web Application (CSWA)
Local Network
Fog Node
Database
Fog Trust Anchor (FTA)
Fog Access Control Agent (FACA)
Fog Controller
RabbitMQ
Fog Trust Provider (FTP)
Connectivity Context-Awareness Agent (CCAA)
Behavior Context-Awareness Agent (BCAA)
Location Context-Awareness Agent (LCAA)
Light Sensor
Temperature Sensor
Humidity Sensor

Figure 4.1: Software architecture

4.3 Components Deployment

FC per definition aims at a services deployment model that relies on creating a highly virtualized services execution environment, improving the utilization of existing hardware devices in the network and efficiency of the deployed resources. FNs are envisioned as services execution hosts, which offer required computational, storage, and networking capabilities for deployed services. As described in Section 3.1.3, two mainstream approaches for virtual execution environments are VMs and containers. Compared to VMs, containers introduce performance benefits, requiring fewer resources for their execution. Since IoT devices are resource-constrained and lightweight execution is favored, the chosen service deployment model in the scope of this dissertation relies on containers.

Docker[36] is a container-based platform, offering support for application development, deployment, and execution. Developed and supported by large developers and companies community, Docker provides a multitude of open-source solutions for deploying services and applications on a variety of devices (e.g., desktop computers, single-board computers, and Cloud Servers). By separating services execution from the infrastructure using containers, Docker enables a simple services deployment and services portability, since they do not depend on the underlying operating system or installed software stacks. Docker containers are deployed and installed as Docker images. Images are downloaded during the installation from Docker Hub[37] and afterwards executed on the host. Through the simple installation and deployment, Docker offers services scalability and load-balancing capabilities, allowing services to scale their performance depending on the computational capabilities of the hosting devices.

In order to deploy the developed CC and FC services on FN using Docker, respective container images have to be created and uploaded to the Docker Hub. Docker images creation relies on the derivation of new images from the existing ones, which are already uploaded to the Docker Hub. As stated in Section 4.1, the developed services rely on the Spring Boot framework, which encapsulates the developed service's business logic in a single Java Archive (.jar) file. Thus, the creation of a Docker image results in deriving an existing, open-source Java-supporting Docker image by installing a compiled .jar file that contains the service's business logic.

36 `https://www.docker.com/`, last access May 2, 2022
37 `https://hub.docker.com/`, last access May 2, 2022

```
1 FROM arm32v7/openjdk:11-ea-11-jre-slim
2 ARG \${JAR_FILE}=target/*.jar
3 COPY \${JAR_FILE} faca.jar
4 RUN chmod +x /faca.jar
5 ENTRYPOINT ["java","-jar","/faca.jar"]
```

Listing 4.1: Service container image compilation

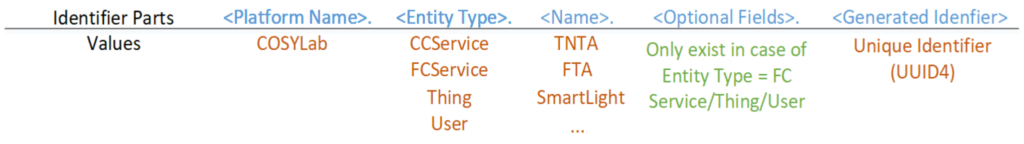
An example of Docker image creation is presented in Listing 4.1, showing the image creation steps for the FACA service. The original Docker image, from which the new image is derived, is specified at line 1. Specification and installation of the compiled .jar file into the newly created Docker image are configured in lines 2-4. Afterwards, the generated image's entry point, defined through a command that will start a service execution, is presented in line 5. Using this command, the compiled .jar file will be launched on the Docker image startup, leading to the execution of the developed service. Once the image creation configuration is done, the image building process is started through the command `docker build . -t IMAGE_NAME`, where `IMAGE_NAME` stands for the name of the developed service, i.e. FACA or FTA.

Once the Docker images are created and uploaded to the Docker Hub, their deployment to FNs and Cloud Server is possible. Listings A.2, A.3, and A.4 in Appendix A present service configuration bundles for FC services. Furthermore, Listing A.1 describes the service bundle that is deployed on Cloud Server. Configuration bundles are deployed using the Docker's extension for a multi-container environment's deployment - Docker Compose[38].

4.4 Trust Management

The implementation for TN mechanisms in the scope of this dissertation relies on a hierarchical trust model, utilizing PKI and digital certificates to establish trust connections between FC services, Things, and users (cf. Section 3.2). The implementation of TN software components: TNTA, FTA, and FTP enables the trust management encapsulation within FC-based TNDs, independently of the Cloud Servers availability. Moreover, an extension of the implemented PKI schema through security profiles offered by the FTP component improves the proposed solution's applicability on resource-constrained Things. The upcoming Sections 4.4.1 - 4.4.3 provide implementation and technical details considering TN deployment and provisioning.

38 https://docs.docker.com/compose/, last access May 2, 2022



Identifier Parts	<Platform Name>.	<Entity Type>.	<Name>.	<Optional Fields>.	<Generated Idenfier>
Values	COSYLab	CCService	TNTA	Only exist in case of	Unique Identifier
		FCService	FTA	Entity Type = FC	(UUID4)
		Thing	SmartLight	Service/Thing/User	
		User	...		

Figure 4.2: Identifier Structure

4.4.1 *Identity & Certificate Management*

Following the identity naming structure proposed in Section 3.2.1, identifiers for four different entity types present in the COSYlab are built. Identifiers for each entity type follow the structure provided in Figure 4.2, containing several dot-separated identifier parts. All identifiers contain the following common characteristics: (1) each identifier has the IoT platform name "COSYLab" as its first part, (2) the second part represents the entity type : `CCService`, `FCService`, `User`, or `Thing`, (3) the entity name is provided as the third part, and (4) each identifier contains a generated identifier at its end, which is created using Universally Unique Identifier [LMS05]. Other identifier parts are specific to the each entity type and contained as optional fields. Based on the entity type, optional fields within the identifier can have the following values:

- The FC Service entity has the FN identifier,
- The Thing entity FN identifier and FTP component identifier, and
- The User entity has the FN identifier and FACA component identifier as optional fields.

The values contained in the optional fields are used to describe the associations between entities in the Cloud - Fog - Thing hierarchy, following the schema presented in Figure 3.4. For that, all optional fields contain the FN identifier, indicating the device where IdP (FTP or FACA component) is hosted. Things' and Users' identifiers additionally contain the FTP's or FACA's unique identifier, which uniquely defines the deployed component that issued their identity.

Managing trust in the proposed CTFTP and FTTTP requires issuing digital certificates depending on the entity type: CC service, FC service, User, and Thing. Issued certificates contain general information on the certificate holder, as well as constraints and certificate validation endpoints specific to each entity type.

```
 1 Certificate:
 2  Data:
 3   Version: 3 (0x2)
 4   Serial Number: 1621897045653 (0x179a098e695)
 5  Signature Algorithm: ecdsa-with-SHA256
 6  Issuer:
 7   countryName = AT
 8   organizationName = COSYLab
 9   commonName = COSYLab.CCService.TNTA.37a71ae4-14f6-4950-a028-af9f8b9ac5fd
10  Validity
11   Not Before: May 24 22:57:25 2021 GMT
12   Not After : May 24 22:57:25 2022 GMT
13  Subject:
14   countryName = AT
15   organizationName = COSYLab
16   commonName = COSYLab.FCService.FTA.d81f20bf-f644-4ba7-a21f-a508e2b0f435
17               .fde93417-d0ff-4f50-a658-85bfb89dcf07
18  Subject Public Key Info:
19   Public Key Algorithm: id-ecPublicKey, Public-Key: (256 bit)
20    pub:
21     04:bb:52:f7:e6:27:25:fe:20:ed:14:a5:d5:a4:3a:
22     a5:ec:26:de:b5:e2:d5:da:b4:1a:0c:33:1c:7a:8b:
23     ab:b8:c7:98:05:11:9a:8f:fa:ca:82:07:24:44:ef:
24     f6:68:bc:ba:5b:47:f2:ff:89:fd:0d:fb:26:59:a4:
25     9f:fc:94:0d:e7
26   ASN1 OID: prime256v1
27  Signature Algorithm: ecdsa-with-SHA256
28   30:46:02:21:00:e8:c7:82:cd:4a:7c:a5:5f:ff:1a:dd:5f:9d:
29   85:95:45:25:ba:1d:ad:d2:ba:6c:18:54:2f:82:a6:fd:21:5c:
30   63:02:21:00:98:64:91:30:2e:5d:7a:d8:a2:e9:90:5f:44:c9:
31   41:21:11:74:5f:40:98:7c:d1:7e:ea:d7:c3:91:a3:41:79:40
```

Listing 4.2: Digital certificate content

```
 1 X509v3 Key Usage: critical
 2  Digital Signature, Non Repudiation, Key Encipherment, Certificate Sign,
 3 CRL Sign
 4  X509v3 Basic Constraints: critical
 5 CA:TRUE, pathlen:3
 6  X509v3 CRL Distribution Points: critical
 7 Full Name:
 8  URI:http://ni-bakk.cosy.univie.ac.at/tnta/fog/certificate/list/revoked
 9 X509v3 Authority Key Identifier:
10  URI:COSYLab.CCService.TNTA.37a71ae4-14f6-4950-a028-af9f8b9ac5fd
11  serial:01:79:A0:39:75:38
```

Listing 4.3: TNTA Certificate Extensions

General certificate information is common to all entity types and is presented in Listing 4.2. Each certificate contains fields required by the X.509 version 3 standard: version, serial number, and signature algorithm (cf. lines 3-5). Based on the PKI performance evaluation through simulation and the retrieved results presented in Section 5.2, the signature algorithm ecdsa-with-SHA256 has been chosen for the developed solution as the best performing option. Issuer and subject (certificate holder) information describe their country, organization (IoT platform), and their common name, mapped to the identity naming scheme, as presented in lines 6-9 and 13-17, respectively. Additionally, the certificates contain the subject's public key (lines 18-26) and the issuer's digital signature (lines 27-31). The certificate's validity period is limited with start and end moments, as documents in lines 11 and 12, granting trust to the subject for the specified period.

Specific capabilities for each entity type are described through X.509 version 3 extensions. Applied extensions are:

- *Key Usage* limits certificate usage to particular use cases,
- *Basic Constraints* defines if the subject can act as CA and how long the trust chain can be,
- *CRL Distribution Points* enable certificate revocation check in the CTFTP,
- *Authority Key Identifier* defines which certificate should be used to verify CRL's digital signature, and
- *Authority Information Access* specifies the OCSP endpoint in the FTTTP.

As described in Section 3.2.2, CTFTP contains two CC services and FTA as FTTTP's gateway, with the TNTA service as the IoT platform's root CA. For that reason, TNTA's certificate extensions shown in Listing 4.3 enable the TNTA's public key pair to be applied for digital signing, encryption, and decryption information as well as issuing certificates and managing CRL (lines 2-5). Furthermore, as TNTA issues the digital certificates for FTA instance, it manages CRL for all FTAs that joined CTFTP and exposes the API for retrieving CRL (lines 6-8), whose authenticity can be verified using the TNTA's public key defined in lines 9-11.

FTA represents the bridge between CTFTP and FTTTP. Depending on the availability of TNTA service, FTA can hold two different types of the digital certificate (cf. Listing 4.4). If FTA has never been connected to the TNTA, it uses a self-signed certificate that is valid in the FTTTP, and its revocation is validated using the OCSP endpoint listed in lines 13-14. Otherwise, FTA obtains its certificate from TNTA, and its revocation is verified using TNTA's API, as enlisted in lines 6-11. In both cases, FTA acts as a root CA for the whole FTTTP (lines 1-4), allowing FTA's public key pair to be applied for signing information (e.g., Tickets or OCSP responses) and issuing digital certificates to other FTTTP entities.

FACA and FTP also represent FTTTP CAs as formulated in Listing 4.5, allowing them to issue certificates (lines 1-4) to users and Things, respectively. Since they exist in the FTTTP and obtain their certificates from FTA, the OCSP endpoint hosted within FTA and exposed using AMQP protocol is applied for verifying if their certificate has been revoked, as described in lines 5-6.

```
X509v3 Key Usage: critical
 Digital Signature, Non Repudiation, Certificate Sign
X509v3 Basic Constraints: critical
 CA:TRUE, pathlen:2
// When issued by TNTA
X509v3 CRL Distribution Points: critical
 Full Name:
  URI:http://ni-bakk.cosy.univie.ac.at/tnta/fog/certificate/list/revoked
X509v3 Authority Key Identifier:
 URI:COSYLab.CCService.TNTA.37a71ae4-14f6-4950-a028-af9f8b9ac5fd
 serial:01:79:A0:39:75:38
// When self-signed certificate
Authority Information Access: critical
 OCSP - URI:amqp://fta.rpc.certificate.ocsp
```

Listing 4.4: FTA certificate extensions

```
X509v3 Key Usage: critical
 Digital Signature, Non Repudiation, Certificate Sign
X509v3 Basic Constraints: critical
 CA:TRUE, pathlen:1
Authority Information Access: critical
 OCSP - URI:amqp://fta.rpc.certificate.ocsp
```

Listing 4.5: FACA and FTP certificate extensions

```
1  X509v3 Key Usage: critical
2   Digital Signature, Non Repudiation
3  X509v3 Basic Constraints: critical
4   CA:FALSE
5  Authority Information Access: critical
6   OCSP - URI:amqp://fta.rpc.certificate.ocsp
```

Listing 4.6: C-A agents certificate extensions

C-A agents, users, and Things represent leaf nodes in the FTTTP. For this reason, their certificates' extensions (X509v3 Basic Constraints and Key Usage) deny them the possibility to issue other certificates. C-A agents' certificates, as presented in Listing 4.6, allow C-A agents to digitally sign messages (lines 1-4) sent to the FTA and FACA (cf. Section 3.4.1), proving their authenticity. Since C-A agents' certificates are issued only by FTA, their revocation can be validated using the already mentioned OCSP endpoint (lines 5-6).

Users' and Things' certificates contain the same extension sets, as presented in Listings 4.7 and 4.8, respectively. Their certificates are applied for digitally signing information and mutually authenticating with other FTTTP entities. Furthermore, based on the Data Encipherment in the Key Usage extension, users and Things can apply their public key for encrypting exchanged information. However, as presented in line 2 of Listing 4.8, Things can also use their certificate to perform a key exchange and perform a mutual authentication in the machine-to-machine communication pattern. Finally, lines 6-7 in both listings define OCSP endpoints for issuing CAs - FACA for users and FTP for Things.

```
X509v3 Key Usage: critical
 Digital Signature, Non Repudiation, Data Encipherment
X509v3 Basic Constraints: critical
 CA:FALSE
Authority Information Access: critical
 OCSP - URI:amqp://faca.rpc.pip.ocsp
```

Listing 4.7: User certificate extensions

```
X509v3 Key Usage: critical
 Digital Signature, Non Repudiation, Data Encipherment, Key Agreement
X509v3 Basic Constraints: critical
 CA:FALSE
Authority Information Access: critical
 OCSP - URI:amqp://ftp.rpc.certificate.ocsp
```

Listing 4.8: Thing certificate extensions

4.4.2 *Trust Bootstrapping & Management*

The proposed PKI schema, along with the identity naming scheme and the digital certificates described in the previous section, allows IoT entities to mutually authenticate themselves and establish trust relationships.

The first step for distributing trust from CC to FC services is for the FTA to join CTFTP by obtaining its certificate from TNTA. As presented in Figure 3.6, this is based on offline-shared secrets: *Application Secret* and *Instance Secret*. These secrets are deployed with the FTA's docker image and entered in the TNTA's database to whitelist the given FTA. Listing 4.9 provides an example for the whitelisted FTA secrets (lines 2-3) and their SHA-256 hash as presented in line 4.

The first step's result is FTA's successful registration at TNTA, meaning that FTA obtained its certificate and is recognized as trustworthy in the CTFTP. TNTA stores the information on all registered FTAs in its database, as presented in Listing 4.10. This information contains FTA's identity (lines 2-3), its digital certificate in the Privacy Enhanced Mail (PEM) format (lines 4-21), and the CSR token (line 22) that has been exchanged during the first step for securely handling FTA's secrets. Finally, information on the registered FTA contains timestamps for: (i) when FTA has registered (line 23) and (ii) when FTA's certificate has been revoked (line 24).

```
1 { "_id" : ObjectId("60b1710f8666c10da0d0c605"),
2  "applicationSecret" : "Rq3by#K0l!lvoanp57hg",
3  "instanceSecret" : "0U^8HLzBS1kNaK1QacHm",
4  "hash" : "880548be6e9465529766eb732fd7c6acaa7a0e4babf0e09fd41f16079779cbc2" }
```

Listing 4.9: FTA whitelist credentials object

```
{"_id" : ObjectId("60b1710f8666c10da0d0c607"),
 "identity" : "COSYLab.FCService.FTA.eb3e3e09-be23-4f2a-96ae-3475211dfe0d
            .69f0e0a5-d4c6-4cb2-8481-24e489b45d72",
 "certificate" : "-----BEGIN CERTIFICATE-----
  \nMIIC3TCCAoKgAwIBAgIGAXm1IabeMAoGCCqGSM49BAMCMGUxRDBCBgNVBAMMOONP
  \nU1lMYWIuQ0NTZXJ2aWNlLlROVEEuMzdhNzFhZTQtMTRmNi00OTUwLWEwMjgtYWY5
  \nZjhiOWFjNWZkMRAwDgYDVQQKDAdDT1NZTGFiMQswCQYDVQQGEwJBVDAeFw0yMTA1
  \nMjgyMjM5MTJaFw0yMjA1MjgyMjM5MTJaMIGJMWgwZgYDVQQDDF9DT1NZTGFiLkZD
  \nU2VydmljZS5GVEEuZWIzZTNlMDktYmUyMy00ZjJhLTk2YWUtMzQ3NTIxMWRmZTBk
  \nLjY5ZjBlMGE1LWQ0YzYtNGNiMi04NDgxLTI0ZTQ4OWI0NWQ3MjEQMA4GA1UECgwH
  \nQ09TWUxhYjELMAkGA1UEBhMCQVQwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAAS9
  \ndAWpNxDGFKSQX9MG0q/W6C9Q8NAb4fqd4aLyEHNZ3o0lDG8FCxPb/JkExmlH14d+
  \nj9XZ3GMhYp1zxmfz5kkdo4H4MIH1MA4GA1UdDwEB/wQEAwICxDASBgNVHRMBAf8E
  \nCDAGAQH/AgECMDgGA1UdHwEB/wQuMCwwKqAooCaGJGh0dHA6Ly9uaS1iYWtrLmNv
  \nc3kudW5pdmllLmFjLmF0L2ZvZzBDBggrBgEFBQcBAQEB/wQ0MDIwMAYIKwYBBQUH
  \nMAGGJGh0dHA6Ly9uaS1iYWtrLmNvc3kudW5pdmllLmFjLmF0L2ZvZzBQBgNVHSME
  \nSTBHoT2GOONPU1lMYWIuQ0NTZXJ2aWNlLlROVEEuMzdhNzFhZTQtMTRmNi00OTUw
  \nLWEwMjgtYWY5ZjhiOWFjNWZkggYBeaA5dTgwCgYIKoZIzj0EAwIDSQAwRgIhAI2I
  \nkwogbjVsuklZpUWlCoxUfWYJzT1R4fBolB3xfJP2AiEA84DkIE+3x9RGTGC8rZ8K
  \nP1Pvc8oUMbRBMGQ+xVSjoys=
  \n-----END CERTIFICATE-----\n",
 "csrToken" : "7c79949a-72c5-45c4-9c11-944bbee86821",
 "registeredAt" : ISODate("2021-05-28T22:39:11.804Z"),
 "certificateRevokedAt": null}
```

Listing 4.10: FTA registered certificate object

In the second step, FTA initializes FTTTP and issues certificates to the other FC services (FACA, FTP or C-A agents) upon their startup. The issued certificates are stored in FTA's database following the object structure presented in Listing 4.11. Similar to TNTA, FTA stores FC service information about its identity and certificate content (lines 3-16), as well as certificate creation and revocation timestamps (lines 17-18). Compared to TNTA, FTA is missing the CSR token field since FC service registration occurs in a local network, often on the same device. Due to that, certificate issuing message exchanges can be protected using network isolation mechanisms (e.g., firewalls or AMQP broker AC).

Trust bootstrapping ends with the second step, meaning that all CC and FC services can establish trust relationships using their certificates. From that point on, to authenticate themselves, services attach *Tickets* once accessing another service API. Tickets are attached as message headers: HTTP in CTFTP and AMQP in FTTTP, and sent to the receiving service. They represent a JSON Web Signature (JWS) token [JBS15], signed with the Ticket issuer's private key, enabling digital signature and identity verification at the receiving entity.

The Ticket's example is provided in Listing 4.12. Ticket header (line 1) and signature (lines 14-15) contain information on the applied cryptographic operations to protect the Ticket's integrity. The Ticket's body contains general information: the validity period (lines 9-10) and issuance details (lines 11-12). Furthermore, the Ticket's body contains claims for the mutual authentication in services:

```
{
 "_id" : ObjectId("60a12effeb73dc0400322a9f"),
 "identity" : "COSYLab.FCService.FTP.a01398ce-1383-4dae-b8b8-59a57341608b
           .0f403fa7-505b-45d8-96d6-efe3e8ccaf54",
 "certificate" : "-----BEGIN CERTIFICATE-----
  \nMIIB3zCCAYWgAwIBAgIGAXl1n5UFMAoGCCqGSM49BAMCMGoxaDBmBgNVBAMMX0NP
  \nU1lMYWIuRkNTZXJ2aWNlLkZUQS5hMDEzOThjZS0xMzgzLTRkYWUtYjhiOC01OWE1
  \nNzM0MTYwOGIuZTcxZTdlM2QtMTU5NC00MGIzLTg0MzEtMjVjODRmYmU5ZWVjMB4X
  \nDTIxMDUxNjE0NDEwM1oXDTIyMDUxNjE0NDEwM1owajFoMGYGA1UEAwxfQ09TWUxh
  \nYi5GQ1NlcnZpY2UuRlRQLmEwMTM5OGNlLTEzODMtNGRhZS1iOGI4LTU5YTU3MzQx
  \nNjA4Yi4wZjQwM2ZhNy01MDViLTQ1ZDgtOTZkNi1lZmUzZThjY2FmNTQwWTATBgcq
  \nhkjOPQIBBggqhkjOPQMBBwNCAAQKtAkplo8ZYz+/JB/RHWNFBjLW14dTMToHLkQQ
  \n4nu5DQKt9QNMMOgDnfo8IMOkjQbGQSoBhkbLZTPeZc/RBoc6gQIEwKMTMBEwDwYD
  \nVR0TBAgwBgEB/wIBADAKBggqhkjOPQQDAgNIADBFAiEAnCT+GR9kqmi/UiY/9bL8
  \nQYrJ1A/Z4E9zFHfcddJnhHUCIEEYZ6ghj3yLLdPWbAZrJAFySGQ/rqINvl9m53+I
  \nVbF1\n-----END CERTIFICATE-----\n",
 "registeredAt" : ISODate("2021-05-16T14:41:03.239Z"),
 "certificateRevokedAt": null
}
```

Listing 4.11: FC service registered certificate object

```
{ "alg": "ES256" }
{
 "ticketType": "REQUEST",
 "msgHash": "e0541b29df2f66742aded5b613bb12ce6a4cb6762d21c611bc58c7e7154a3399",
 "functionality": "/certificate/list/revoked",
 "iss": "COSYLab.FCService.FTA.d81f20bf-f644-4ba7-a21f-a508e2b0f435
      .fde93417-d0ff-4f50-a658-85bfb89dcf07",
 "sub": "COSYLab.CCService.TNTA.37a71ae4-14f6-4950-a028-af9f8b9ac5fd",
 "exp": 1622162596,
 "nbf": 1622155396,
 "iat": 1622155396,
 "jti": "103981d9-6480-4238-8d8a-f3565776465e"
}
Signature: BDr8HdjnMIDQPf_gD3Encn1EgqnpFnJPDPjPA86MbFWDy6jqdlv5qOJV
         GbdBgAXBc8uRgGO-dyVrWpbbd0xYeA
```

Listing 4.12: Ticket example

1. *ticketType* describes the message type: request or response,
2. *msgHash* holds the SHA-256 hash of the original message, hence disabling unauthorized message alterations during transmission,
3. *functionality* defines the API endpoint for which the Ticket is issued and valid,
4. *iss* specifies the Ticket issuer, which digitally signed the Ticket, and
5. *sub* defines the Ticket's subject - service for which the Ticket is created.

Upon obtaining the Ticket, the invoked service validates the issuer's digital signature, proving the Ticket's and issuer's authenticity. Afterwards, the service checks message integrity using the *msgHash*. Finally, the service checks if the *ticketType* and *functionality* match the invoked service API and verifies the Ticket's time constraints. If all these verifications succeed, the received message is considered trustworthy and the service proceeds with the message processing.

The following trust management step considers certificate revocation. The critical point in this step is the certificate revocation notification to other TND entities. As described in Section 3.2.2, it is decided to apply CRL in CTFTP and OCSP in FTTTP. For that reason, for FTAs' certificate revocation, TNTA provides an endpoint that allows fetching CRL information. Listing 4.13 gives details on the CRL object content. Namely, the CRL object contains FTA's identity and revoked certificate serial number (line 1-3). Also, minor meta-information about revocation is provided, indicating that the certificate is no longer valid (line 4) and the revocation timestamp (line 5).

The applied OCSP approach in FTTTP allows checking certificate revocation at the CA service that initially issued the certificate. The entity that verifies the certificate queries the CA service and obtains information on the certificate's validity. As presented in Listing 4.14, response to the OCSP query contains identification data for the certificate holder and the certificate itself (lines 1-3). Lastly, the OCSP response contains the current certificate validity status (line 4), which can have the following values: ”good”, ”revoked”, or ”unknown”.

```
1  { identity: "COSYLab.FCService.FTA.cecfb3fe-702a-4e3f-a09a-d4aa91dc4b86
2           .5cdf3a4a-b524-475e-82aa-3a983dbdfd6e",
3   certificateSerialNumber: 1621897045653,
4   isValid: false,
5   revokedAt: 2021-05-28T17:46:11.438 }
```

Listing 4.13: CRL object example

```
1  { identity: "COSYLab.FCService.CCAA.a01398ce-1383-4dae-b8b8-59a57341608b
2           .d5a76c3a-ed4c-4a10-95f1-15bb6629a08d"
3   certificateSerialNumber: 3671399173341
4   status: "revoked" }
```

Listing 4.14: OCSP response example

4.4.3 *End-to-End Security for Things*

Based on the design for offloading the asymmetric encryption-required operations from Things to FNs described in Section 3.2.3, the ”best effort” model for integrating resource-constrained Things into the proposed trust management schema has been implemented. For that purpose, based on the best security capabilities the Thing can support, Thing and FTP establish a direct trust relationship that is categorized using the security profiles described in Table 3.3. Upon successful trust establishment, FTP takes over the PKI-related cryptographic operations for the Thing and supports a mutual authentication between the Thing and other

IoT network entities. Another feature FTP offers is provisioning CA capabilities for Things that are capable of performing all required PKI operations. In that scenario, FTP issues a certificate (cf. Section 4.4.1) to a Thing, and the Thing manages standalone its trust relationships during its lifecycle.

```
 1 {
 2  "_id" : ObjectId("60a164a026eeff51ff67931b"),
 3  "registeredId" : "CosySenc_1",
 4  "keyExchanged" : true,
 5  "securityProfile" : "SYMMETRIC_SECURE_KEY",
 6  "keyAlias" : "cosySenc_1_symm_key",
 7  "generatedId" : "COSYLab.Thing.Smart Light.a01398ce-1383-4dae-b8b8-59a57341608b
 8                 .0f403fa7-505b-45d8-96d6-efe3e8ccaf54
 9                 .16ccdc94-7229-4724-8570-d8440ca60291",
10  "deviceType" : "Smart Light"
11 }
```

Listing 4.15: FTP Thing credentials object

Offloading PKI-related operations to FTP requires managing the Things' credentials. Credentials management is exposed through the FTP API, resulting in storing the Thing's information in the FTP's database. This information contains relevant data to define the Thing in the PKI schema (cf. Listing 4.15). First of all, the Thing's data object contains information obtained during the trust bootstrapping process depicted in Figure 3.8: line 3 contains the Thing's default identity, line 4 determines if trust is established through a key exchange, line 5 defines the applied security profile, and line 6 documents the pointer to the exchanged key in FTP's key storage. PKI-related information contains the Thing's identity generated by FTP (lines 7-9) and the Thing's device type (line 10).

After successfully establishing a direct trust with FTP, the Thing relies on FTP's computational capabilities to perform PKI-related operations. Messages produced by the Thing are digitally signed by FTP and forwarded to other IoT entities. To achieve this, the Thing measures and sends information to FTP, securing it using security credentials established during the trust bootstrapping process. Upon receiving a message, FTP checks if the credentials correspond to the Thing and tries decrypting the message. Once the decryption is finished, FTP generates JWS containing the original message (cf. Listing 4.16) and meta information about the Thing so that other TND entities can then prove message's and Thing's authenticity. Finally, FTP returns JWS to the Thing so that the Thing can forward JWS to other nodes in the TND for further processing.

```
1  {
2    "alg": "ES256"
3  }
4  {
5   "functionality": "readDimLevel",
6   "message": "{dimmingLevel: 70}",
7   "securityProfile": "SYMMETRIC_SECURE_KEY",
8   "iss": "COSYLab.FCService.FTP.a01398ce-1383-4dae-b8b8-59a57341608b
9        .0f403fa7-505b-45d8-96d6-efe3e8ccaf54",
10  "sub": "COSYLab.Thing.Smart Light.a01398ce-1383-4dae-b8b8-59a57341608b
11       .0f403fa7-505b-45d8-96d6-efe3e8ccaf54
12       .de20d80c-df9d-4344-a29f-dcf9a3e4c795"
13  "exp": 1621196963,
14  "nbf": 1621189763,
15  "iat": 1621189763,
16  "jti": "907d627f-16c6-4851-8684-367ca4dd7336"
17 }
18 Signature: Kpk4jvprYCkhYn7o5VY6_XbGUCd78oGja8n5GyR5IVa5KcUxtcMRf115vQD1
19          Fw6SwhFJx5J7zv9hbdEEMPE2ow
```

Listing 4.16: FTP signed message example

Listing 4.16 presents JWS example that is issued by FTP to the Thing. Besides the standard JWS header (lines 1-3) and signature (lines 18-19), the JWS body contains Thing- and trust-relevant information. General JWS data, such as time information and the unique JWS identifier, are enlisted in lines 13-16. Trust-relevant information is represented through the issuer's (FTP's) and Thing's identity (lines 8-12), as well as the security profile the Thing uses (line 7). Finally, the original message information is described through the invoked Thing's functionality and the produced message content, as presented in lines 5 and 6, respectively.

4.5 PKI Scalability Simulations

As described in the Section 3.2.2, IdMS's efficiency and scalability play a significant role in their adoption in IoT systems. To evaluate the applicability of different IdM and KMP approaches, a simulation environment has been designed and implemented. The main goal of the simulations was to evaluate the impact of PKI solutions on the computational and storage resources, as well as the latency in an IoT network.

Existing simulation tools offer various features for the evaluation of network and data security properties of IT systems. Concerning data security, several simulators offer capabilities for simulating various network attacks. Nessi2[39]

39 http://www.nessi2.de/, last access May 2, 2022

and Cymulate[40] incorporate capabilities for simulating real cyber-attacks based on detected security gaps. Moreover, in the IoT security area VANETsim[41] and NetSim[42] offer simple ways to verify the establishment of security concepts in IoT systems. However, despite being categorized as data security simulators, all the simulators mentioned above do not serve the intended purpose for IdM and KMP performance simulations.

For that reason, the implemented simulator is based on the NS3[43] network simulator. NS3 offers capabilities for emulating data exchanges between network entities in both IP and non-IP based networks, accompanied by a variety of network protocols like Wi-Fi [EA13] and LTE [Koo11]. Additionally, IdM and KMP properties relevant to IMDS simulation are integrated through NS3 plugins. Further implementation details for IdM and KMP properties are provided in Sections 4.5.1 and 4.5.2.

4.5.1 *Simulator Setup*

The implemented network simulator plugins enable the creation of a hierarchical network environment, representing CAs and Things. For that, three types of network nodes have been modeled:

1. *Root Nodes* representing the root CA in the system;
2. *Intermediate Nodes* representing the intermediate CAs between the root CA and the Things;
3. *Leaf Nodes* representing resource-constrained Things.

The Root Node represents the root CA in the system, issuing and managing digital certificates for the Intermediate Nodes. Also, it is assumed that the Root Node has the most computation power in the network. Intermediate Nodes manage the complete identity lifecycle for Leaf Nodes by creating, validating, expiring, and revoking Leaf Nodes' certificates. The envisioned computation power of Intermediate Nodes is less than the one of the Root Node, but higher than the Leaf Nodes' computation power. Leaf Nodes represent single Things in the system. They have the least computation power of all the nodes and can only manage their own certificate. That means that a Leaf Node cannot create, validate, expire or revoke certificates of other nodes in the network.

40 https://cymulate.com/, last access May 2, 2022
41 https://svs.informatik.uni-hamburg.de/vanet/index.html, last access May 2, 2022
42 https://www.tetcos.com/, last access May 2, 2022
43 https://www.nsnam.org/, last access May 2, 2022

The identity lifecycles are modeled through the workflows of nodes and KMPs and predetermined in the simulation through the configuration properties. *Lifecycles* are configured through JSON-formatted instructions for the simulator to run the necessary steps to simulate certificate creation, validation, and revocation. Instructions are defined through built-in *Cycles*:

- The *Create-L Cycle* causes the creation of a certificate for the Leaf Nodes;
- The *Create-I Cycle* creates a certificate for the Intermediate Nodes;
- The *Validate Cycle* executes the certificate validation procedure for establishing communication between two Leaf Nodes;
- The *Expire-Revoke Cycle* revokes digital certificates on the nodes, based on time or detected security breach.

The steps that need to be executed by a node to complete particular Cycle are modeled through *Tasks*. Tasks define the amount of computational, storage, and time resources required to finish an IdM- and KMP-related operation. Each Task can have different computation requirements, and these requirements can be different for each node. For example, creating a certificate takes longer in an Intermediate Node than in the Root Node. Tasks are separated into two groups: (1) NodeTask and (2) WaitTask. A NodeTask simulates a workload that is executed on a node. Each node is featured with a queue containing NodeTasks, which are processed in a serialized order based on the availability of the processing threads on the node. A WaitTask represents the operation that is executed on another node, i.e. Leaf Node requests signing a public key from the Intermediate Node. If a node starts a request, it waits for the response on that request to complete the WaitTask. To model the request failures due to the network and processing timeouts, each WaitTask has a maximum waiting time, upon which expiration the WaitTask is considered as failed.

Scalability evaluation scenarios are described using models for Cycles and Tasks. Key factors for describing scenarios, as described in Section 3.2.2, are: (1), the asymmetric encryption type, (2) the digital certificate type (3), the key exchange protocol, and (4) the digital certificate revocation schemes. By combining the key factors, the following simulation scenarios are described:

- Encryption Type: RSA, Certificate Type: X.509, Key Exchange Protocol: Diffie-Hellman,
- Encryption Type: EC, Certificate Type: X.509, Key Exchange Protocol: Diffie-Hellman, and
- Encryption Type: EC, Certificate Type: Implicit, Key Exchange Protocol: Qu-Vanstone.

Furthermore, support for the evaluation of digital certificate revocation schemes has been implemented. OCSP works with queries and runs on the Root Node and Intermediate Nodes. As the standard map's query complexity is modelled using Red-Black Trees[44], a single data read from a map is expressed as `O(log(n))`, where `n` is the size of the map. In contrast, CRL iterates through a revocation list of entries to determine if the certificate has been revoked. Therefore, the time required for CRL-based certificate validation corresponds to the size of the revocation list `n`. Revoked certificates are stored as entries consisting of the revoked certificate serial name, as well as the date of revocation.

The ultimate goal for building the PKI simulator was to evaluate the following performance metrics while executing different IdM and KMP protocols: CPU usage, storage consumption, as well as the network transmission and processing latency. CPU usage of the node is abstracted through the workload percentage for executing Tasks. Each Task is described via the processing time it requires to complete. Moreover, for each node, the idle time is measured, i.e. the time during which no Task is executed. Through that, the workload percentage over the nodes' simulated lifetime is calculated for each node as the sum of the workloads of the Task processing and idle times, divided by the total simulation time. Moreover, the processing threads number in each node is considered, so that the workload of a Task consumes a fraction of CPU usage, expressed as `100%/overall_number_of_threads`. Finally, CPU usage is expressed using the Equation 4.1.

$$CPU\_Usage = \frac{\sum_{t=1}^{T_{simulation}} \frac{Tasks(t)}{Total\_Threads_{node}}}{T_{simulation}} \tag{4.1}$$

The processing and network transmission latency expresses the required time for the execution of a single Cycle. For a Cycle to complete, all defined steps containing Tasks have to be executed. Each NodeTask is configured with the time it requires to execute, which corresponds to the Task's processing latency. A WaitTask involves sending a request to another node and waiting for the response, incorporating processing and network transmission latency. Hence, WaitTask's processing latency represents processing latency on another node, while WaitTask's network transmission latency is calculated as `WaitTask_Execution_Time - Another_Node_Processing_Latency`.

44 https://www.usna.edu/Users/cs/crabbe/SI321/current/red-black/red-black.html, last access May 2, 2022

For storage consumption, measurements of the size of the encryption keys, digital certificates, and identities are traced during and after each Task execution, so that storage requirements in each node can be simulated concerning IdM and KMP operations. Furthermore, the occupation of storage influences the duration of the Task's execution. Namely, if a node accesses its storage location to fetch a digital certificate, or to validate a CRL list, the Computation Time (CT) is calculated using Equation 4.2. According to this formula, CT equals close to the default Computation Time (CTD) if the memory is barely used, meaning that finding desired storage location is straightforward. In case that memory is more used, finding desired storage location is harder. For that reason, CT increases for the proportion of used storage, which is derived from the Maximum Available Memory (MAM) and the Current Memory Usage (CMU).

$$CT = CTD * (1 + \frac{CMU}{MAM}) \tag{4.2}$$

4.5.2 *Simulator Configuration*

The NS3 extensions mentioned in Section 4.5.1 are implemented using the C++ language and integrated into the NS3 engine through plugins. The plugins represent NS3-applications, embodying nodes in the PKI-based network. Moreover, both simulation and node configuration are highly customizable so that various settings of PKI-systems can be simulated by adjusting the simulator configuration properties. The configuration properties are formatted using the JSON format and loaded as files into the simulator once the simulation execution starts. For the NS3 simulator to run, certain events in the system have to be defined, triggering actions at a given time. These actions incorporate Cycles, executing series of Tasks. In the developed simulator, Cycles are randomly started on each node, describing the dynamic nature of IoT network - nodes joining and leaving the network, validating each others' certificates, and revoking certificates in case of a security breach or certificate validity expiration.

Nodes are implemented through three classes: RootNode, INode, and LNode[45], supporting separated operations as described in Section 4.5.1. Still, several configuration properties (cf. Table 4.1) are defined to specify their behavior during

45 For the sake of more understandable configuration properties, the term *Parent Node* is defined, describing the higher-level node in PKI system that handles digital certificates for a particular node (e.g., Root Node for Intermediate Node and Intermediate Node for Leaf Node).

Table 4.1: Simulator configuration properties - nodes

Property Name	Description
max_thread	Gives the node a number of simulated threads. It determines how many NTasks the node can finish in a single loop.
max_storage	Defines the maximum storage the node has in bytes.
max_ram	Defines the maximum of RAM the node has in bytes.
max_tries	Defines the amount of time a WaitTask can be retried before a Lead Node searches for a new Intermediate Node or the Intermediate Node declares that the Root Node does no longer respond.
wtask_seed_lower	Defines the lower boundary for selecting random moment when the node retries a WaitTask in milliseconds.
wtask_seed_upper	Defines the upper boundary for selecting random moment when the node retries a WaitTask in milliseconds.
validate_lower	Defines the lower boundary for selecting random moment when nodes initiate the next validation circle.
validate_upper	Defines the upper boundary for selecting random moment when nodes initiate the next validation circle.
revoke_lower	Defines the lower boundary for selecting random moment when the Root Node revokes randomly chosen Leaf Node.
revoke_upper	Defines the upper boundary for selecting random moment when the Root Node revokes randomly chosen Leaf Node.

Table 4.2: Simulator recipient identifiers

Recipient identifier	Description
a	Send to the affected Leaf Node. The affected node is used in the validation cycle between a starter leaf node and an affected leaf node.
t	Does not send to another node but to this node.
r	The Root Node. Used if you need direct communication with the root node from the leaf node.
s	Sends the package back to the node that requested Task execution.
p	Sends the package to the parent of the node.

Table 4.3: Simulator configuration properties - steps

Property Name	Description
name	Unique name of step.
ram_strain	Defines how much heap RAM is used by this step.
time	Defines the required duration for the step execution.
check_valid_aff	The step checks whether the parent node has a valid certificate. This is used to stop the cycle if, for example, a CRL entry states that the certificate has been revoked.
check_revoke_list	Adding times required for simulate time required for validating certificate revocation using CRL or OCSP.
payload.size	Defines how big the dead payload should be. The dead payload is sent with the package but is not used afterwards by PKI operations, allowing simulations of bigger network packages.
payload.mult_with	Defines a multiplier for the dead payload. For example, if a CRL List is sent it needs to be multiplied with the number of already revoked certificates.
next_step	Defines the next step that follows the current step. It is split between multiple options. Each option defines nextStep (unique identifier of the next step in the Cycle) and sendTo properties (cf. Table 4.2).

the simulation, triggering actions in the NS3 simulator. For the time-triggered actions, the upper and lower boundary for selecting random moments are configured, resulting in the definition of the upper and lower boundaries for triggering *Validate Cycles* and *Expire-Revoke Cycles*. Furthermore, network transmission timeouts, as well as processing overloads in nodes are considered. For that purpose, the upper and lower boundary for requests retransmission and the maximum number of retransmission before considering Task as failed are added in configuration properties. Finally, nodes' computational capabilities are configurable by setting nodes' processing threads number and memory and storage capabilities.

Configuration properties for Cycles and Tasks are provided in Table 4.3. Each Task is defined through its name, memory consumption, and the time it needs to complete. The Task's duration is directly mapped to the consumption of one node's thread for the given time, therefore indicating CPU usage. Further properties define dependencies on PKI-operations in the defined Task's scope of, i.e. checking the parent's node certificate validity and if a certificate is revoked. Each Task indicates the Cycle's transition to the next Task the node executes or which node is the recipient of the Task's result (in case of a WaitTask). The configuration property values for recipients are defined in Table 4.2.

Besides the nodes and Tasks configuration, general simulation properties describing characteristics of IdM and KMP protocols in PKI scope are defined (cf. Table 4.4). These properties allow defining storage requirements for encryption keys, digital certificates, and the size of the CRL and OCSP entries. Moreover, to simulate the confidentiality of the communication links based on the asymmetric encryption-based key distribution (e.g., TLS or DTLS), symmetric encryption keys re-exchange frequency can be configured, therefore reducing or increasing time intervals between symmetric keys exchange.

Once nodes, Tasks, IdM, and KMP properties are configured, general properties on simulation execution are yet to be provided as input arguments for NS3 simulator application. These properties involve: (1) frequency for checking nodes with the possibility of triggering PKI-related operations, (2) simulation duration, (3) number of Intermediate and Leaf Nodes in the simulated network, and (4) latency and bandwidth of the network.

Table 4.4: Simulator configuration properties - global

Property Name	Description
name	KMP and IdM schema that is being simulated
crl_entry_size	Defines the size of a single CRL/OSCP list item.
asym_key_size	Defines the size of the asymmetric encryption key.
sym_key_size	Defines the size of the symmetric encryption key.
session_key_recheck	Defines how long it takes to recheck the validation of already established secured links between Leaf Nodes using symmetric encryption keys.
cert_size	Defines the size of the digital certificate.

4.6 Access Control Implementation

As justified in the Section 3.3, the AC deployment, and its application in the scope of this dissertation aim at utilizing FC services to distribute AC to the edge of the network, allowing the low latency and integration of context information into the access policies validation. For that, AC modules are deployed on FNs and can operate independently on CC availability. Moreover, ABAC-based fine-grained access policies allow the security policy definition based on IoT users' generic attributes, as well as context information sensed in the IoT environment. The upcoming Sections 4.6.1 - 4.6.3 provide implementation and technical details considering AC deployment and provisioning.

4.6.1 *Access Control Components*

The components for AC provisioning, ACAM and FACA, enable deployment of security policies on FNs while maintaining consistency of the security policies and the trust relationship between CC- and FC-based services. As illustrated in Figure 4.1, the ACAM represents the central entity in the AC system, to which all FACAs are connected. The ACAM's primary purpose is to maintain a registry of the deployed FACAs, as well as overall security policies and IoT device types in the IoT system. Therefore, ACAM provides the initial setup for the FACAs and the configuration during their operation. Despite its central role in the system, a constant up-time of the ACAM is not mandatory, since FACAs are designed and developed to provide AC services independently of the ACAM's availability.

Maintained registries incorporated in ACAM are stored in the database, as depicted in Figure 4.3. The `DeviceType` table holds entries of IoT device types, specified through the device type name, device vendor (service provider), firmware versions, and supported functionalities. Device type entries are managed by IoT system administrators and applied to FACA through the routines for the synchronization of the access policies' configuration, described in Section 4.6.3. Device type functionalities are managed through `DeviceVersion` table since this table stores updates on IoT device functionalities, accompanied by a version identifier, timestamp, and a human-readable changelog. IoT functionality changes are stored in `Change` table contain the functionalities update for each version. Changes are then forwarded from ACAM to FACAs during the synchronization of the access policies' configuration and applied to access policies deployed in FACAs.

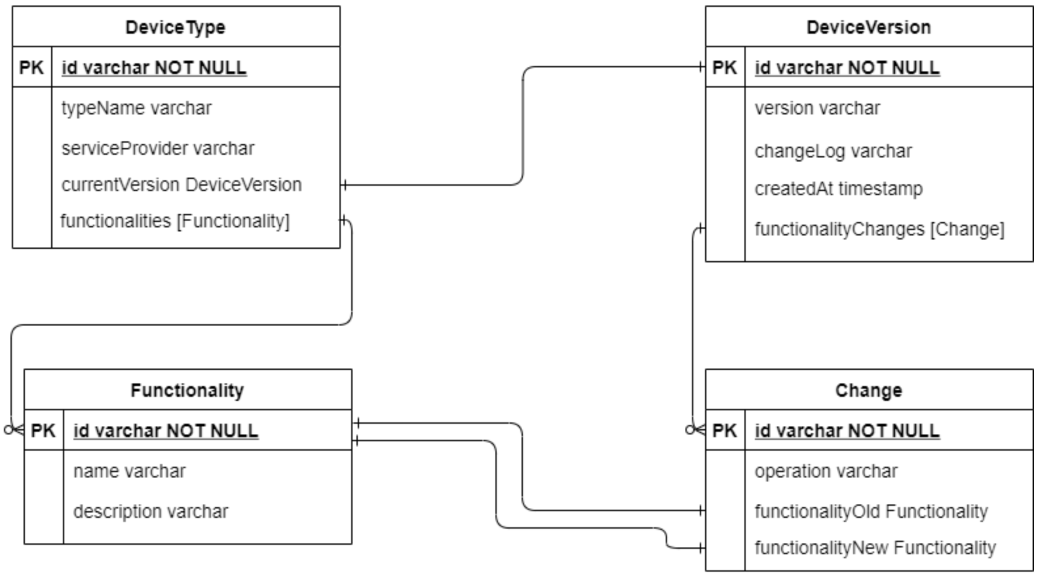


Figure 4.3: ACAM database schema

The FACA component offers authentication, authorization, and access policy management services to the components deployed at the IoT network's edge. It is deployed on FN and built based on ABAC using XACML (cf. Section 4.6.2). As discussed in Section 3.3.2, the central endpoint for the user management in the proposed system is located within the FACA, offering the user authentication and access rights authorization independently of ACAM's availability. User information is stored within the FACA's database in the `subject` table presented in Figure 4.4. This table contains fields for:

- User authentication containing username, password hash, password salt, and digital certificate;
- User account management that involves information like user role, if the user account is activated, or if the account is blocked;
- User information that is mapped to attributes and afterward used for authorization, i.e. profession, birth date, handicaps.

Based on the information in the `Subject` table, the user is able to authenticate and obtain a session token, which is used to authorize user actions in the IoT system. To enable ABAC-based AC, an issued session token contains all user's attributes and is valid for a predefined period. Attributes are built based on predefined user attributes (cf. Table 4.5) and user information from table `Subject`. Once the session token has expired, it needs to be reissued through the renewed user authentication. To incorporate user attributes in a session token and to

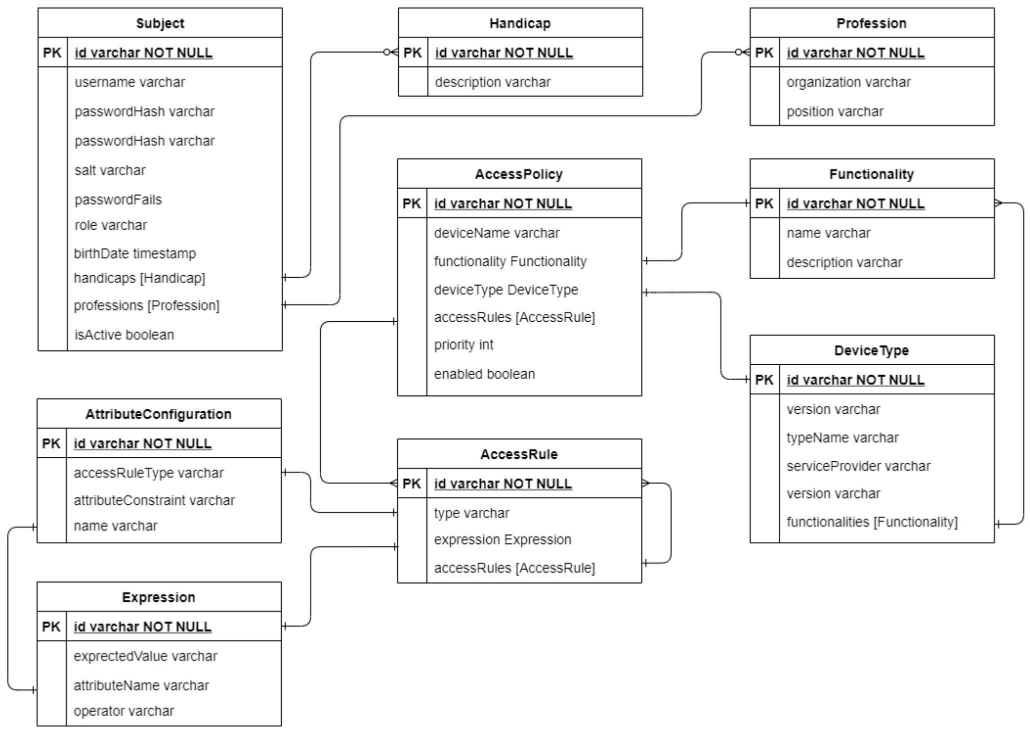


Figure 4.4: FACA database schema

protect the session token's integrity using FACA's digital signature, JWS is chosen as a session token format. Once issued, the JWS token can be presented by the user to FACA while trying to access IoT service. FACA will evaluate if the access can be granted based on the attributes from the JWS token (cf. Listing 4.17).

The JWS token contains digitally signed information on a user and session. It is encoded using Base64 [Jos06] and separated into three parts: header, body, and signature. Listing 4.17 presents the header and body of the JWS token since the signature contains non-readable information. The JWS token is based on user attributes presented in Table 4.5 and the identity naming scheme described in Section 3.2.1. The header is shown in line 1-3, defining the asymmetric key encryption algorithm used for signing the JWS token - ES256 [Jon15]. Lines 5-10 present user information which are relevant for IoT service access validation. Lines 11-15 contain identity information on token issues and token holder, holding identities of FACA component and a user authenticated at FACA. Session token duration and validity information is listed in lines 16-18, defining details on when the token expires, since when the token is valid, and when the token is issued. Line 19 presents the JWS token identifier.

```
{
   "alg":"ES256"
}
{
   "Subject.Identity.Username":"nignjatov",
   "Subject.Person.Handicap":"["Shortsightedness"]",
   "Subject.Work.Profession":"["Research Assistant"]",
   "Subject.Work.Organization":"["University of Vienna"]",
   "Subject.Identity.Role":"Owner",
   "Subject.Person.Age":29,
   "iss":"COSYLab.FCService.FACA.20769199-d9dc-4f98-aae3-f81388636852.
         66f2019e-93d3-4940-93f4-e6b6615ca0fd",
   "sub":"COSYLab.User.20769199-d9dc-4f98-aae3-f81388636852.
         66f2019e-93d3-4940-93f4-e6b6615ca0fd.
         fd1d7b8c-4368-4a82-acc4-88728d06b661",
   "exp":1598456159,
   "nbf":1598448959,
   "iat":1598448959,
   "jti":"f2624a44-be0f-48ca-affa-03f7d0f4c4ab"
}
```

Listing 4.17: JWS session token example

```
1 String id = "b69d68bf-8f25-4334-af21-dfb92e87237a";
2 String name = "Subject.Identity.Role";
3 AccessRuleType accessRuleType = AccessRuleType.STRING;
4 AttributeConstraint constraint = new AttributeConstraint();
5 constraint.setAllowedValues(["OWNER","ADMINISTRATOR","DEVICE_MANAGER","GUEST"]);
```

Listing 4.18: Role attribute value configuration

Besides user management, FACA enables management support for access policies. FACA manages information on registered IoT device types and their functionalities, as presented in the database tables in Figure 4.4. This information is initially retrieved from ACAM and updated through the synchronization procedures for the access policy configuration described in Section 4.6.3. Fields in the `AccessPolicy` table affected by this synchronization procedures are: (1) `enabled`, mapping presence of the IoT device's functionality in the configuration defined in ACAM, and (2) `Functionality` that tracks IoT device's functionality name across multiple IoT device's configuration versions in ACAM. Furthermore, FACA-enabled access policy management relies on IoT device functionalities as the PA. Afterwards, attributes and their expected values are bound to PA for a successful IoT device functionality authorization. Attributes present in access policies are configured through `AttributeConfiguration` table. This table contains constraints on attributes that have to be respected for the attribute used in any access policy. Constraints depend on the attribute value's data type (number, text, enumeration) and access rule type for the selected data type, as depicted in Section 4.6.2. An attribute constraint example is presented in Listing 4.18, declaring attribute constraint identifier at line 1, attribute name at line 2, access rule type at line 3, and attribute's list of allowed values in lines 4 and 5.

Table 4.5: User attributes

Attributes Name	Description
Subject.Identity.Username	User’s account name
Subject.Identity.Role	User’s role
Subject.Person.Age	User’s age
Subject.Person.Handicap	User’s handicap
Subject.Work.Organization	User’s company
Subject.Work.Profession	User’s profession

4.6.2 *Attribute-Based Access Policies Management*

The implementation of fine-grained access policies involves the application of ABAC and XACML standards. The utilization of ABAC enables the usage of generic properties as input for the access policy validation. Therefore, the IoT service access authorization implemented in the scope of this dissertation relies on attributes from two sources: (1) user information and (2) context information (cf. Section 4.7). To classify attributes according to their sources, dot-delimited attribute naming schema is used. The first two parts of each attribute name indicate the attribute source class and subclass, such as Subject.Identity or Context.Behavior, while further attribute name parts are custom to the components that define attributes, i.e. FACA, BCAA, LCAA, and CCAA. Since FACA manages user information, all user-related attributes implemented in the proposed solution are managed by FACA as given in Table 4.5. Further context-related attributes are documented in Sections 4.7.1 to 4.7.2, based on localization, user behavior, and system connectivity context information, respectively.

Access policy management and enforcement in FACA is highly influenced by the usage of XACML as the underlying standard, affecting the modeling and application of access policies. This enforces the decomposition of the FACA services into four modules: (1) PIP offers services required for management of the user attributes, IoT devices functionalities supported by IoT devices, as well as handling the session management (cf. Section 4.6.1), (2) PAP enables the definition and management of access policies, (3) PDP evaluates them in order to grant or deny access to IoT devices, and (4) PEP defines an interface required for the authorization of access requests and propagates them further to PDP. The services offered by these modules are exposed through external HTTP and AMQP interfaces in the FACA, providing other system components the required AC functionalities.

The access policy management implemented in PAP offers CRUD operations on access policies and access rules. The tables `access_rule` and `access_policy` presented in Figure 4.4 utilize IoT device functionalities and attributes configuration as an input for building access policies. Access policies incorporate one or multiple access rules and are bound to PA. Furthermore, access policies grouping around the principles presented in Section 3.3.1 are implemented through the access policy priority, which describes the importance of an access policy, i.e. if an access policy can override the authorization decision of other access policy in case of an authorization decision conflict. Priorities (`0` is the most important) represent access policy grouping principles as follows:

- The value `0` represents an access policy bound directly to the particular IoT device functionality,
- The value `1` represents an access policy bound to all functionalities of the particular IoT device,
- The value `2` represents an access policy bound directly to the particular functionality of all IoT device with same device type, and
- The value `3` represents an access policy bound to all functionalities of all IoT devices with the same device type.

Access rules types define attribute data types and possible expressions that PDP can evaluate during the authorization. The implemented access rule types are:

- The type `NUMERIC` enables comparison of number-based attribute values and expected access policy values;
- The type `STRING` enables comparison of text-based attribute values and expected access policy values;
- The type `BOOLEAN` enables validation if attribute value is true or false;
- The type `COMPOSITE` allows binding other access rules through logical operators, leading to access rules nesting and complex access policies, consisting of multiple access rules.

Listing 4.19 presents an access policy stored in the FACA's database in JSON format. As shown in lines 2-5, the access policy contains its unique database identifier, the access policy priority, and PA defined through device name and function, respectively. Furthermore, lines 6-28 illustrate access rules for the access policy, allowing access to the IoT service for *a person that is older than 18 and studies at the university.* Access is permitted through three access rules incorporated into a `COMPOSITE` type (line 7) and bound using the AND operator (line 8), whereas each rule defines its type, attribute name, operator, and expected value.

PDP implements the authorization logic based on access rule types. This logic is embedded into evaluation expressions which are bound to the each access rule. An expression is defined through: (1) evaluation operator (e.g., equals, contains, or greater than), (2) name of the attribute for which the value will be evaluated, and (3) expected attribute value. Evaluation operators are implemented based on the access rule type. As an example, Listing 4.20 presents an enumerated operators list for the text-based access rules.

```
1  {
2      "_id" : ObjectId("5f45050898a6b90ed03636ba"),
3      "deviceName" : "Living Room Light Sensor",
4      "function" : "readValue",
5      "priority" : 0,
6      "rule" : {
7          "accessRuleType" : "COMPOSITE",
8          "operator" : "AND",
9          "accessRules" : [
10             {
11                 "accessRuleType" : "NUMERIC",
12                 "expectedValue" : 18.0,
13                 "attributeName" : "Subject.Person.Age",
14                 "operator" : "GREATER_THAN",
15             },
16             {
17                 "attributeName" : "Subject.Work.Organization",
18                 "expectedValue" : "University",
19                 "operator" : "CONTAINS",
20                 "accessRuleType" : "STRING"
21             },
22             {
23                 "attributeName" : "Subject.Work.Profession",
24                 "expectedValue" : "Student",
25                 "operator" : "EQUALS",
26                 "accessRuleType" : "STRING"
27             }
28         ],
29     }
30 }
```

Listing 4.19: Access policy example

```
public enum StringRelationalOperator {
    EQUALS,
    EQUALS_IGNORE_CASE,
    CONTAINS,
    CONTAINS_IGNORE_CASE,
    NOT_CONTAINS,
    NOT_CONTAINS_IGNORE_CASE,
    STARTS_WITH,
    STARTS_WITH_IGNORE_CASE,
    ENDS_WITH,
    ENDS_WITH_IGNORE_CASE
}
```

Listing 4.20: Text access rule operators

Once the authorization starts, PDP executes the expression evaluation by presenting a session token to the access rule evaluator. Based on the predefined attribute name, the evaluator extracts the attribute value and compares it with the expected attribute value based on the configured operator. Finally, the evaluator returns the true or false value, indicating if the authorization has been successful. The evaluator for text-based access rules is presented in Listing 4.21. Based on the selected operator at line 3, the switch-case selects the required branch (e.g., at line 7 or line 9) and compares expected and provided attribute value.

```
1  boolean evaluateStringExpression(String expectedVal, String controlledVal,
2          StringRelationalOperator operator) {
3    switch (operator) {
4      case EQUALS:
5        return (expectedVal.equals(controlledVal)) ? true : false;
6      case EQUALS_IGNORE_CASE:
7        return (expectedVal.equalsIgnoreCase(controlledVal)) ? true : false;
8      case CONTAINS:
9        return (expectedVal.contains(controlledVal)) ? true : false;
10     case CONTAINS_IGNORE_CASE:
11       return (expectedVal.toLowerCase().contains(controlledVal.toLowerCase()))
12                             ? true : false;
13     case NOT_CONTAINS:
14       return (!expectedVal.contains(controlledVal)) ? true : false;
15     case NOT_CONTAINS_IGNORE_CASE:
16       return (!expectedVal.toLowerCase().contains(controlledVal.toLowerCase()))
17                             ? true : false;
18     case STARTS_WITH:
19       return (controlledVal.startsWith(expectedVal)) ? true : false;
20     case STARTS_WITH_IGNORE_CASE:
21       return (controlledVal.toLowerCase().startsWith(expectedVal.toLowerCase()))
22                             ? true : false;
23     case ENDS_WITH:
24       return (controlledVal.endsWith(expectedVal)) ? true : false;
25     case ENDS_WITH_IGNORE_CASE:
26       return (controlledVal.toLowerCase().endsWith(expectedVal.toLowerCase()))
27                             ? true : false;
28     default:
29       return false;
30     }
31 }
```

Listing 4.21: Text access rule expression evaluation

4.6.3 *Access Control Distribution*

The AC service distribution in this dissertation's scope is enabled through the deployment of FACA components on FNs. To achieve that, the trustworthiness of the deployed FACA components and security policy consistency have to be ensured, which is achieved through two approaches. The first approach is based on the overall TN provisioning principles for FC services, as described in Section 4.4.2. The application of these principles allows the authentication of FACA instances within the COSYLab. Therefore, services of FACA (e.g., session token issuance or access authorization) can be uniquely identified and validated for their trustworthiness.

The second approach focuses on ensuring the trust of FACA's authorization capabilities through the configuration of access policies from ACAM. As documented in Section 3.3.2, FACA is capable of deriving local access policies based on the access policy configuration provided by ACAM and adapting its access policies during the system's runtime, allowing the possibility of disabling access to IoT services in case of a security breach. This is achieved through whitelisting of Things' functionalities to enable or disable the provisioning of IoT services through FACA.

Since AC management in FACA relies heavily on the Things' functionality as PA, synchronization techniques for the distribution of the supported Things' functionalities had to be employed. In order to tackle this requirement, the solution relies on the firmware life-cycle versioning of the individual Things, offering a mapping of functionalities based on a particular firmware installed on them. Enlisted functionalities (e.g., turn on the light or read room's temperature) are exposed to the IoT system's administrators for adding, updating, or removing a particular functionality. Once updated, functionalities are stored in ACAM's database in the tables `device_type`, `functionality`, `functionality_change`, and `device_version` (cf. Figure 4.3) and exposed to FACAs through the REST interface. Afterwards, the Things' functionalities and firmware versions are retrieved by FACAs, resulting in adjusting the predefined access policies in the FACA and managing the particular IoT service to the end-users.

FACA fetches the access policy configuration from ACAM, which contains the whitelisted Things' functionalities. Fetching configuration occurs periodically (e.g., every 3 hours) or event-based (e.g., in case of suspicious behavior in IoT environment or acknowledged security breach). The implicit requirement for access policy configuration synchronization is the possibility for FACA to establish the network connection towards ACAM. Once FACA retrieves the access policy configuration and detects a new firmware version for one or multiple Things, it aligns the entries in the `device_type` and `functionality` database tables (cf. Figure 4.4). Through the alignment, the functionalities for the particular device type can be enabled, disabled, or updated. Since the `functionality` table's entries are used as PA in `access_policy` database table, any changes of functionalities directly affect FACA's access policies, therefore enabling or disabling access to IoT services.

Since the alignment of access policies has to keep track of changes imposed by the configuration and functionalities supported by a particular firmware version on the Thing, a simple versioning system for functionalities is designed and developed. The firmware version is described through a version number, a log of changes provided in the versions, as well as a timestamp, allowing to track the history of versions. Moreover, each version describes changes that are enforced on the services of the Thing. These changes are documented through `actions` that can be separated into:

1. `CREATE` for creation of a new service of a Thing;
2. `CHANGE` for update on the existing service;
3. `REMOVE` for deletion of the existing service.

Listing 4.22 provides an example for the access policies configuration. In line 1 the unique identifier of the Thing's device type is given, followed by information of the firmware in lines 3-8. Available functionalities are coded in lines 9-19.

```
1  "deviceId": "Smart Light",
2  "firmware": {
3    "version": {
4      "versionNumber": "1.1",
5      "timestamp": 1555982664905,
6      "changelog": "Added change color.
7      Renamed turnOn function to turnOnLight."
8    },
9    "functionalities": [{
10     "machineName": "turnOnLight",
11     "humanReadableName": "Turn on the light",
12     "action": "CHANGE",
13     "changeFrom": "turnOn"
14   },
15   {
16     "machineName": "changeColor",
17     "humanReadableName": "Change light's color",
18     "action": "CREATE"
19   }]
20 }
```

Listing 4.22: Access policy configuration version example

4.7 Context-Aware Access Control

The designed C-A AC solution developed in this dissertation's scope allows the integration of context information and its utilization in security policies, as argued in Section 3.4. Designed interfaces and data models allow the straightforward extension of AC mechanisms through various C-A agents. This leads to the FACA's capability to collect and manage context information and offer users the possibility to create AC policies based on C-A attributes and utilize them during the authorization. The following Sections 4.7.1 - 4.7.3 provide implementation details on the C-A AC extensions, enabling the integration of context information into the AC mechanisms described in Section 4.6 using C-A AC API.

4.7.1 *Context-Awareness Components Integration*

Context information exchange between C-A agents and FACA occurs through the C-A AC API, as described in Section 3.4.1. Using C-A AC API, C-A agents can register themselves, configure context information they observe as C-A attributes, and distribute C-A attribute values during their lifecycle. This section describes implementation details concerning the C-A AC API message exchanges during the C-A agent's operational phase, as illustrated in Figure 3.11.

The first step in the C-A agent's operational phase is the C-A attributes registration (cf. Listing 4.23), describing the context information C-A agent will be delivering, as well as how FACA can apply it. Thereby `contextType` defines the type of context the C-A agent supports (line 3). The C-A agent's attribute value publishing strategy is defined through the field `distributionPattern` (line 5), `valueMaximumTimePeriodSeconds` expresses the maximum time distance allowed to occur between two attribute value notifications (line 6), and `attributeValueValiditySeconds` (line 7) describes for how many seconds FACA should consider a notified value valid. Based on the registered attributes, FACA allows security policy creation. For that purpose, C-A agents register a unique `attributeName` (line 2), `accessRuleType` (line 4) stating which access rule type will the attribute support (cf. Section 4.6.2), and the range in which certainty on notified attribute values will reside (lines 8-11). Finally, to enable synchronous attribute values retrieval during the authorization, the C-A agent can register the AMQP route using the `attributeEvaluationRoute` field, which FACA invokes during the authorization of critical operations (cf. Section 4.7.3).

```
{
   "attributeName":"Context.Connectivity.ConnectedToCloud",
   "contextType":"CONNECTIVITY",
   "accessRuleType":"BOOLEAN",
   "distributionPattern":"PERIODIC",
   "valueMaximumTimePeriodSeconds":10,
   "attributeValueValiditySeconds":100,
   "certaintyRange":{
      "minimum":0.0,
      "maximum":100.0
   },
   "attributeEvaluationRoute":null
}
```

Listing 4.23: C-A attribute registration

```
{
   "createAttributes":[
      {
         "attributeName":"Context.Connectivity.PingOK",
         "contextType":"CONNECTIVITY",
         "accessRuleType":"BOOLEAN",
         "constraint":null,
         "distributionPattern":"PERIODIC",
         "valueMaximumTimePeriodSeconds":10,
         "attributeValueValiditySeconds":100,
         "certaintyRange":{
            "minimum":0.0,
            "maximum":100.0
         },
         "attributeEvaluationRoute":null
      }
   ],
         "updateAttributes":[
      {
         "oldAttributeName":"Context.Connectivity.PingOK",
         "newAttributeName":"Context.Connectivity.IsPingOK"
      }
   ],
   "deleteAttributes":[
      "Context.Connectivity.IsPingOK"
   ]
}
```

Listing 4.24: C-A attribute configuration update

```
{
   "attributeName":"Context.Connectivity.ConnectedToCloud",
         "contextType":"CONNECTIVITY",
   "attributeValue":"true",
   "certainty":80.0,
   "timestamp":[2021,3,30,18,57,35,834955400]
}
```

Listing 4.25: C-A attribute value notification

Furthermore, the C-A agent can update its attributes configuration anytime during its lifecycle once it detects new context information that might be usable for AC mechanisms. For that purpose, the C-A agent updates the C-A attributes configuration in FACA (cf. Listing 4.24). The configuration update can occur through three different operations: create, update (rename), and delete. The create operation (lines 2-17) follows the same approach as the C-A attributes registration described in the previous paragraph. The update operation presented in lines 18-23 allows changing registered `attributeName`, whereas delete operation (lines 24-26) completely removes C-A attribute configuration. Still, update and delete operations affect the security policy and notified attribute values management, using the following strategies:

- Renaming the C-A attribute results in aligning all access policies containing access rules with the renamed C-A attribute to use the new attribute name. All notified attribute values are updated to the new attribute name.
- Deleting the C-A attribute leads to disabling all access policies containing that C-A attribute.

Once the C-A attribute is configured at FACA, the C-A agent sends attribute value notifications as presented in Listing 4.25. Each value notification identifies the targeted C-A attribute through `attributeName` and `contextType` fields (lines 2-3). The rest of the notification describes the C-A attribute value through its certainty, sampling moment, and value, contained in the fields `certainty`, `timestamp`, and `attributeValue` (lines 4-6), respectively.

An essential aspect of the C-A AC API is the `attributeName` fields structure. Namely, to support the detection and revocation of multiple context information sources and reduce the C-A attribute management overhead, attribute names (cf. Table 4.6) contain static and dynamic parts, separated by the character ”?”. C-A agents register only the attribute name’s static part, which declares the general context information type it provides. Still, the dynamic part is used in attribute value notifications to identify which IoT entity is reflected through the context information. For example, BCAA monitors present users’ behavior in IoT systems. However, the number of users can be highly dynamic due to the users joining and leaving the network. Thus, BCAA configures the C-A attribute named `"Context.Behavior.Pattern.Device.Single.ControlBased"`, and notified C-A attribute values are using the attribute name `"Context.Behavior.Pattern.Device.Single.ControlBased?userId=nignjatov"`, stating that the notified value is for the behavior pattern analysis of the user with username ”nignjatov”. This allows BCAA to configure the C-A attribute only once and its configuration can be reused for all users that are using the IoT system.

Table 4.6: C-A attributes

Attribute Name	Description
Context.Connectivity.ConnectedToCloud	Indication for successful connection to Cloud Server.
Context.Behavior.Pattern.Device.Single.ControlBased	User behaviour pattern based on a single Thing control actions.
Context.Behavior.Pattern.Device.Single.MonitoringBased	User behaviour pattern based on notified sensor data.
Context.Behavior.Pattern.Device.Group.ControlBased	User behaviour pattern based on a correlated control actions from multiple, grouped Things.
Context.Location.Position.User	User's current position in a Smart Home.

4.7.2 *Context-Awareness Attributes Management*

The integration of context information in access policies through the C-A AC API requires the extension of the FACA's functionalities, enabling context information usage in defining access policies and applying them during the authorization process. Firstly, FACA's database schema, shown in Figure 4.4, is extended with two new tables, as presented in the Figure 4.5.

The table `context_attribute_configuration` stores information on context attributes registered by the C-A agents using Registration or Configuration Update messages (cf. Section 4.7.1). PAP then uses this information to present them to the users and enables the definition of context-based access policies. The field *attributeName* serves as a key for database entities and uniquely identifies context information. Moreover, this field is the central point for matching notified attribute values and attributes defined in access policies, enabling the context-based authorization. The fields *contextType* and *certaintyRange* further describe the registered C-A attribute. The first field specifies the type of context the C-A agent collects (e.g., location or behavior). QoS aspects are described through the second field, defining the ranges for the certainty C-A agent will attach to the notified context attribute values, i.e. 0-100%.

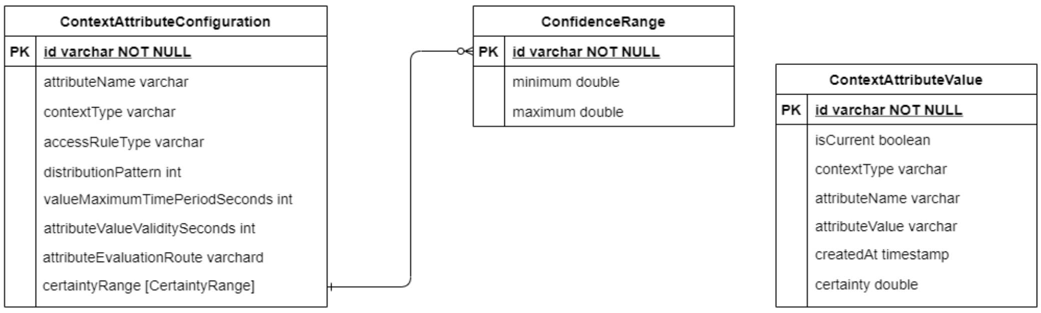


Figure 4.5: C-A attributes management database schema

The fields `distributionPattern`, `valueMaximumTimePeriodSeconds`, and `attributeValueValiditySeconds` are applied for determining if the C-A agent is healthy and to evaluate if the notified attribute value is still valid and should be used during the authorization. In cases when the context information is periodically distributed, FACA distinguishes if the C-A agent sends the attribute value notifications often enough by checking if the value notification arrived in a time difference smaller than `valueMaximumTimePeriodSeconds`. Moreover, `attributeValueValiditySeconds` field is used for validating the notified values' up-to-dateness by comparing the notification's arrival timestamp and the current system time. If the attribute's configured number of seconds is surpassed, the notified value will be discarded and not used during the authorization.

Finally, the fields `accessRuleType` and `attributeEvaluationRoute` impact the access policy definition and validation procedures. The access rule type adheres to the types defined in Section 4.6.2, specifying the type of operations and operators against which attribute values can be validated during the authorization. Since the notified context attribute values can be timely-obsolete with regard to the criticality of the requested operation, as described in Section 3.4.1, C-A attribute values can be fetched directly for C-A agent during the authorization using the API endpoint specified in `attributeEvaluationRoute`, as described in Section 4.7.3. This API endpoint corresponds to the AMQP queue on which the C-A agent exchanges messages during its lifecycle.

Context-based access policies are defined based on the information in the `context_attribute_configuration` table. These policies are stored in FACA's database according to the database schema in Figure 4.4. Listing 4.26 provides an example of such an access policy, where the context-related aspects are contained between lines 10 and 16. The rest of the access policy context conceptually does not differ from the one described in Section 4.6.2. The context-based access rule (line 11) contains two context aspects: value-related and QoS-related. The value-related aspect is documented in lines 12-16, specifying the access rule type, expected value, and attribute name. The QoS aspect of the access policy is described in the `expectedCertaintyRule` object in lines 17-21, enabling the definition of the numeric access rule, specifying the certainty threshold that FACA uses to evaluate if the notified attribute value is valid or not. Lastly, in critical operations, the user can mark the context access rule as time-critical, which will enforce fetching C-A attribute value from the C-A agent during the authorization.

```
1  {
2    "_id" : ObjectId("6067328fcdb7ee1311efd3b0"),
3    "deviceName" : "Living Room Light",
4    "cloudDeviceTypeId" : "5f8600e103031223072051cd",
5    "deviceTypeName" : "Phillips Hue Color",
6    "function" : "turnOn",
7    "rule" : {
8      "accessRuleType" : "COMPOSITE",
9      "accessRules" : [
10      {
11        "accessRuleType" : "CONTEXT",
12        "accessRule" : {
13          "attributeName" : "Context.Connectivity.ConnectedToCloud",
14          "operator" : "IS_TRUE",
15          "accessRuleType" : "BOOLEAN"
16        },
17        "expectedCertaintyRule" : {
18          "accessRuleType" : "NUMERIC",
19          "expectedValue" : 50.0,
20          "operator" : "GREATER_THAN"
21        },
22        "isAttributeValueTimeCritical" : true
23     },
24     {
25        "attributeName" : "Subject.Identity.Username",
26        "expectedValue" : "nignjatov",
27        "operator" : "EQUALS",
28        "accessRuleType" : "STRING"
29     }
30     ],
31     "operator" : "AND"
32   },
33   "priority" : 0,
34   "enabled" : true
35 }
```

Listing 4.26: C-A access policy definition

Once the access policies containing context-related attributes are configured, it is essential to manage the attribute value notifications from the C-A agents. FACA stores notified attribute values in the `context_attribute_value` table. In this table, the notified attribute name, attribute value pairs along with the value's certainty and the notification's timestamp are stored in the fields `attributeName`, `attributeValue`, `certainty`, and `timestamp`, respectively. Afterwards, these fields are used for the authorization purposes, as described in this section and

Section 4.7.3. The field isCurrent marks the last notified value for the particular attribute. Even though the last notified value can be deducted using timestamps, the isCurrent field simplifies database lookup procedures for finding attribute value entries, reducing the processing latency during the authorization.

4.7.3 *Context-Awareness-based Authorization*

Authorization involving C-A attributes contains multiple steps, which increases its complexity in attributes retrieval compared to the authorization based just on user attributes (cf. Figure 4.5). In contrast to collecting attribute values just from the session token in cases when only user attributes are used in an access policy, PDP collects C-A attribute values from multiple sources. Therefore, PDP iterates through the access policy and extracts all the attributes and their access rules that are contained within, checks each attribute's type, and fetches its value as presented in Figure 3.12. Having collected all attribute values, PDP forwards them to the XACML engine for evaluation against the predefined access rules.

The attribute value collection and integration of C-A attributes into the authorization process is presented as pseudo-code in Listing 4.27. In the first stage (lines 1-6), PDP initializes the attribute values list and splits attributes in the access policy to user-related and context-related, whereby context-related ones are separated into critical and noncritical. Afterwards, PDP extracts the user attribute values from the session token if any user-related attribute is found in the access policy as presented in lines 9-11. Subsequently, if the access policy contains noncritical C-A attributes, they are fetched from the database described in Section 4.7.2 as documented in lines 13-17.

In the next stage, PDP checks if critical C-A attributes are in the policy and processes them so that they are grouped into requests sent to C-A agents. Namely, to reduce the number of requests sent to C-A agents and introduced network latency, critical C-A attributes are grouped around common attribute evaluation routes (lines 20-28). This results in listing all C-A attributes with the same configurated evaluation route to be sent in the same request towards the C-A agent. Once all C-A requests are created, they are sent to the C-A agents, and incoming responses are stored in the attribute values list (lines 30-33). Finally, all collected attribute values are forwarded to the XACML engine as presented in line 36, which authorizes the user's access request based on the predefined access policy and collected attribute values.

```
List attrValues = createEmptyList();
// Split attributes from rules in access policy
List userAttrs = extractUserAttributesFromPolicy(policy);
List contextAttrs = extractContextAttributesFromPolicy(policy);
List critContextAttrs = extractCriticalContextAttributes(contextAttrs);
List nonCritContextAttrs = extractNotCriticalContextAttributes(contextAttrs);

// extract user attribute values from session token
if(userAttributes.notEmpty()) {
  attrValues.add(extractUserAttributesFromJWSToken());
}
// fetch non-critical context attribute values from database
if(nonCriticalContextAttributes.notEmpty()) {
  foreach(ncAttr in nonCriticalContextAttributes) {
    attrValues.add(fetchCAAttrValueFromDB(ncAttr));
  }
}
if(criticalContextAttributes.notEmpty()) {
  // create attribute value evaluation requests for C-A agents
  Map caAgentsRequests = createEmptyMap();
  foreach(cAttr in criticalContextAttributes) {
    String evalAttrRoute = fetchEvalAttrRouterFromDBConfig(cAttr);
    if(caAgentsRequests.contains(evalAttrRoute) {
      caAgentsRequests.addRequestItemToRequestsMap(evalAttrRoute,cAttr);
    } else {
      caAgentsReqests.createNewRequest(evalAttrRoute,cAttr);
    }
  }
  // fetch critical attribute values from C-A agents
  foreach(req in caAgentsRequests) {
    attrValues.add(
            amqpClient.fetchAttrValueFromCAAgent(req.evalAttrRoute, req.cAttr));
  }
}
// forward access policy and attribute value to XACML for access validation
return evaluateAccessPolicy(policyData, attrValues);
```

Listing 4.27: Authorization with C-A attributes

5

Evaluation

The implemented COSYLab framework (cf. Section 4) enables TN, AC, and C-A services execution on locally deployed FNs. In order to validate the framework's capabilities and performances, verification scenarios are focused on executing functional and performance evaluations. Evaluations are executed using the COSYLab, relying on the Smart Home scenario described in Section 4.1.

Verification scenarios are divided into the following sections: (1) FC and CC services deployment presented in Section 5.1, (2) PKI performance and efficiency evaluation presented in Section 5.2, (3) TN services deployment presented in Section 5.3, (4) AC services deployment presented in Section 5.4, and (5) Integration of C-A in AC services presented in Section 5.5. Beside these scenarios, the framework's error-handling and recovery procedures concerning the deployed services are evaluated in Section 5.6.

5.1 Fog Services Deployment

Relying on the computational resources of the deployed FNs in IoT environments for the deployment of security services is one of the primary premises for this dissertation. In order to evaluate the feasibility and performances of such an approach, COSYLab CC and FC components are deployed on the Cloud Server, as well as FNs with different computational and storage capabilities, as described in Section 4.1. The evaluation consists of observing the requirements for the

COSYLab components deployment through five aspects: (1) services' installation image size in the Docker container, (2) service startup time, (3) multi-service deployment on a single FN, (4) CPU consumption, and (5) memory consumption during the services execution.

Docker-based CC and FC service deployment requires downloading and installing the Docker images containing executables for the developed components. The developed services (cf. Figure 4.1) and accompanying open-source servers and brokers described in Section 4.1, are deployed using scripts provided in Appendix A. Downloaded Docker images are evaluated to determine storage requirements for hosting COSYLab on Cloud Server and FNs. To achieve that, downloaded image sizes are measured after their installation on the Cloud Server and Raspberry Pi 4 FN device using the `docker image ls` command, and presented in Table 5.1. Open-source services required for COSYLab operations between FC components (AMQP and MongoDB server) are presented as the first two images, followed by the image size of the developed services. Since the developed FC services follow the same Spring Boot-based technology stack, their images have similar sizes, from 244 MB for the Fog Controller up to 297 MB for FACA. Adding all provided image sizes results in 1991 MB of storage required for the initial FC services on a single device. This value increases during the execution of the services since all of them store data in MongoDB. Furthermore, achieving components fail-safety through redundancy increases the storage demand according to the number of the deployed services.

Table 5.1: Docker container image sizes for FC components

Service name	Docker image name	Image size
AMQP broker	rabbitmq	217 MB
Database server	mongo	424 MB
Fog Controller	ignjatov90/fog_ctrl	244 MB
FTA	ignjatov90/fog_fta	270 MB
FACA	ignjatov90/fog_faca	297 MB
CCAA	ignjatov90/fog_ccaa	269 MB
FTP	ignjatov90/fog_ftp	270 MB

Table 5.2: Docker container image sizes for CC components

Service name	Docker image name	Image size
Cloud proxy server	nginx	133 MB
Database server	mongo	449 MB
Cloud configuration server	ignjatov90/cosylab-cloud-config-server	334 MB
Cloud discovery server	ignjatov90/cosylab-cloud-discovery-server	335 MB
TNTA	ignjatov90/cosylab-cloud-tnta	338 MB
ACAM	ignjatov90/cosylab-cloud-acam	340 MB
CSWA	ignjatov90/cosylab-cloud-cswa	134 MB

Deployed CC services' image sizes are presented in the Table 5.2. Compared to the FC services, CC services heavily rely on HTTP communication, resulting in the utilization of Nginx[46] servers instead of RabbitMQ brokers. This also results in bigger CC service images. Despite being based on the same technology stack as the FC components, CC components involve dependency on HTTP servers, increasing the service executables' size. Moreover, FC components' images are derived from the originating Docker images optimized for the Raspberry Pi 4 CPU architecture, resulting in reduced FC components image sizes. In total, the installation of CC components with accompanying servers requires 2064 MB storage space, which is supported by any cloud service provider on the market.

For the second evaluation aspect, to analyze a single FC service execution on an FN device, service startup time is measured. Startup time involves a period from starting the Docker container from the preinstalled image until the component logs print out the information that the service executable is fully deployed. For this purpose, FTP has been taken as a service representative since it involves the complete trust bootstrapping procedure (cf. Figure 3.7) during its startup. Measured startup times for each FN device are:

- Raspberry Pi 4 = 19.41 seconds;
- Raspberry Pi 3 = 33.031 seconds;
- Raspberry Pi Zero = 9538.474 seconds = 159 minutes = 2.65 hours.

46 https://www.nginx.com/, last access May 2, 2022

Startup times indicate that Raspberry Pi 3 and 4 devices can start a single FC service in a reasonable period. A startup time of multiple hours in the case of Raspberry Pi Zero cannot be tolerated in IoT environments since this would mean that the IoT system is not functioning for 2.65 hours once plugged in.

For the third, fourth, and fifth evaluation aspect, a multi-container on single FN device has been set up. Multi-container deployment has been divided through `docker-compose` scripts into three FC service groups: (i) five core docker containers - RabbitMQ, MongoDB, Fog Controller, FTA, and FACA documented in Listing A.2, (ii) FTP service (cf. Listing A.3), and (iii) CCAA as C-A agent representative presented in Listing A.4. For these evaluation aspects the targeted FN devices are Raspberry Pi 4 and Raspberry Pi 3, since Raspberry Pi Zero was not considered eligible for a multi-container deployment due to the long startup time during the second evaluation aspect.

Deployment of the first Docker container group on Raspberry Pi 4 device finished without errors, which further enabled a successful deployment of the remaining FC service groups. However, during the multiple deployments of the first Docker container group on the Raspberry Pi 3 FN, the device became unresponsive, indicating that Raspberry Pi 3 cannot be used for a multi-container deployment. As concluded from the fourth evaluation aspect, the main reason is the lack of RAM in Raspberry Pi 3 device. Namely, multi-container deployment required around 2 GB of RAM, which exceeds Raspberry Pi 3 device's 1GB RAM memory, as documented in Section 4.1. Nevertheless, Raspberry Pi 3 devices can be used for single-container deployments in the COSYLab.

The fourth and fifth evaluation aspects' goals were to measure CPU and memory consumption during the COSYLab FC components usage. For that purpose, *top*[47] and *free*[48] Linux commands, have been used for CPU and memory consumption sampling, respectively. The evaluation duration was 6 hours and it was conducted with all 7 FC services deployed on a single Raspberry Pi 4 FN. To isolate effects of each services on CPU and RAM consumption, the test has been divided in multiple stages, divided by events that represent FC components' activation or deactivation. Stages and their correlation to the enumerated events are listed in Table 5.3. Besides the FC components execution, the impact of the developed FC services on the CPU and RAM consumption has been measured during the whole evaluation by the following COSYLab usage patterns:

47 `https://man7.org/linux/man-pages/man1/top.1.html`, last access May 2, 2022
48 `https://man7.org/linux/man-pages/man1/free.1.html`, last access May 2, 2022

Table 5.3: Docker container resource consumption test stages

Stage name	Executed services	Start event	End event
Setup	Only OS services	Test start	1
Core Containers	Core containers	1	2
All Containers	All containers - TN and AC services idle, C-A usage patterns active	2	3
TN services	All containers - AC services idle, TN and C-A usage patterns active	3	4
AC services	All containers - TN services idle, AC and C-A usage patterns active	5	6
All services	All containers - TN, AC, and C-A usage patterns active	7	8
Shutdown	Only OS services	9	Test end

1. AC services are triggered randomly every 10 seconds by five users requesting access authorization to the IoT device. Each authorization involved validation of 10 access rules in the access policy.
2. C-A services are represented through CCAA, evaluating connection to the Cloud every 10 seconds and sending C-A attribute value to the FACA;
3. TN services are represented through (i) FTA hosting OCSP-based certificate validation for the inter-service communication and (ii) FTP sensor measurements message signing functionalities. In that scenario FTP serves 50 IoT devices with various security profiles: 12 devices with *No Security*, 12 devices with *Integrity*, 13 devices with *Symmetric, Secure Keys*, and 13 devices with *Asymmetric, Secure Raw Public Key* security profile. Each IoT device sent a message signing request every 10 seconds.

CPU consumption measurements for all stages are presented in Figure 5.1, where blue-colored lines and associated numbers indicate the above-mentioned events during the evaluation. Until the first test event, CPU varies around 16%. Afterwards, in the *Core Containers* stage, CPU consumption rises to around 28%. In the remaining evaluation stages, CPU consumption revolves around 30%, except

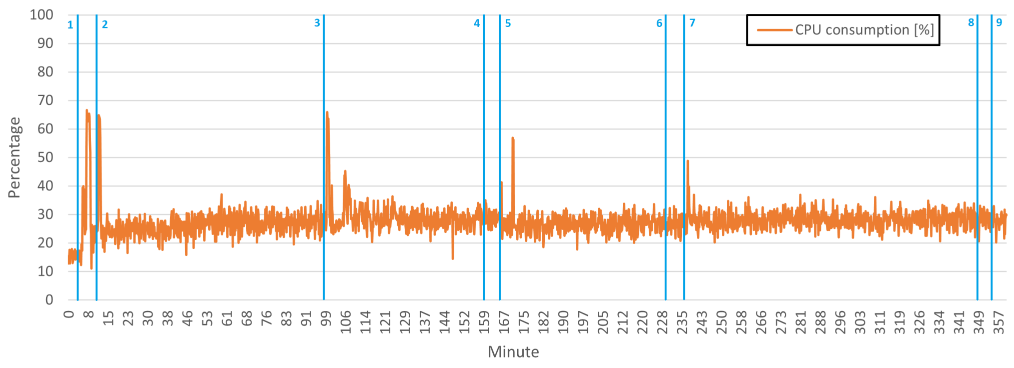


Figure 5.1: FC services Docker container CPU consumption

for the consumption peaks occurring at the start of each further evaluation stage. Still, consumption peaks that have occurred immediately after the new Docker containers are started, that is, after the second and third event, are bigger than the peaks after the fifth and seventh event, where no new containers were started, but different usage patterns have been activated. Overall CPU consumption in stages and consumption peaks indicate that the biggest CPU consumer is the COSYLab services deployment through the Docker containers, whereas services themselves do not significantly impact CPU consumption.

In contrast to the CPU, the fifth evaluation aspect, RAM consumption, shows dependency on the COSYLab services utilization. As presented in Figure 5.2, RAM consumption of 473MB in the *Setup* stage jumps to 1180 MB once *Core containers* stage is entered and 1433 MB after *All containers* has started. Afterward, RAM consumption rises with COSYLab services activation with following values: (a) 1740 MB for *TN services* stage, (b) 1780 MB for *AC services* stage, and (c) 1843 MB for *All services* stage. These values indicate that TN services consumed 21.42% and AC services 24.21% more RAM with active usage patterns. Difference between memory consumption of 1843 MB in *All services* stage and 1433 MB in *All containers* indicates that COSYLab usage patterns execution consumed additional 410 MB RAM which represents an 28.61% increase.

Based on the performed evaluations presented in this section, it can be concluded that FC-based approach is a viable approach for deploying the developed services. First, storage requirements for Docker images installation are satisfied with the most of the SD cards in the market for a couple of euros price. Furthermore, Docker containers can be started on all tested devices, with the present

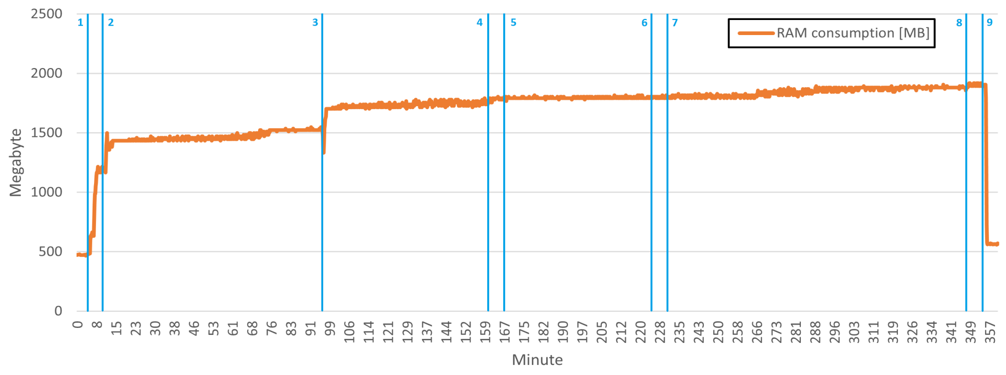


Figure 5.2: FC services Docker container RAM consumption

trade-off between the device price and services startup time. This trade-off is also present in service processing latencies, as further elaborated in Sections 5.3.1 and 5.3.2. Multi-container deployment requires computationally rich devices. To deploy multiple containers on a single device, it is essential to provide sufficient RAM, whereas CPU is less relevant. RAM importance is noticeable from the Raspberry 3 inability to deploy multiple containers since deployment of the COSYLab FC core containers requires more than 1GB RAM.

5.2 PKI Simulations

As the COSYLab aims to provide TN services in IoT networks with resource-constrained Things, the efficiency of the applied KMPs and encryption procedures required to manage the digital certificates (cf. Section 4.4) is critical. The variety of aspects impacting PKI's efficiency, i.e., IoT application domain, network topology, and devices' computational capabilities, dictate a thorough analysis to find the best performing PKI solution for the Smart Home implementation scenario described in Section 4.1. In order to analyze PKI efficiency aspects, simulations have been executed using the PKI simulator described in Section 4.5. Based on the simulation results presented in this section, certificate management procedures described in Section 4.4.1 are implemented and evaluated in Section 5.3.

Simulation goals are to measure CPU and memory requirements for the encryption and certificate management procedures defined in [BD22] - create, validate, and revoke. To achieve that, simulation scenarios provided in Table 5.4 are executed using PKI simulator presented in Section 4.5.1. Configuration parameters

Table 5.4: PKI simulation scenarios

Encryption type	Certificate type	KMP	Certificate revocation
RSA, 3072-bit key	X.509v3	Diffie-Hellman	CRL
RSA, 3072-bit key	X.509v3	Diffie-Hellman	OCSP
EC, 256-bit key	X.509v3	Diffie-Hellman	CRL
EC, 256-bit key	X.509v3	Diffie-Hellman	OCSP
EC, 256-bit key	Implicit	Qu-Vanstone	CRL
EC, 256-bit key	Implicit	Qu-Vanstone	OCSP

for the given scenarios are provided in Appendix B. Key lengths used in simulations follow the security levels published in [BMZ13], aiming to provide a 128-bit security level. Furthermore, to simulate secure communication links between IoT network peers, simulated KMP protocols exchange AES-based session keys, whose validity is limited to 60 seconds.

First, to execute the simulation scenarios, configuration parameters on the used devices need to be provided. To resemble IoT system behavior, following input measurements relevant for the application of PKI-related IdM and KMPs are collected: introduced latencies, computational and storage requirements. Collected measurements are provided as a configuration in Listing B.16 for the devices present in the COSYLab: (i) Raspberry Pi 3 for Root Node, (ii) Raspberry Pi Zero for Intermediate Node, and (iii) NodeMCU ESP32[49] for Leaf Node. Secondly, PKI lifecycles documented in Section 4.5.1 are parametrized using configurations for the above-mentioned PKI simulation scenarios provided in Appendix B with values for the exchanged messages in KMPs, CPU and memory consumption.

The input measurements collection for Raspberry Pi devices is based on the open-source security and cryptography library - openssl[50], using the commands for keys and certificates creation[51] and secrets encryption and decryption. The NodeMCU ESP32 CPU measurements followed the same approach as Raspberry

49 https://nodemcu.readthedocs.io/en/dev-esp32/, last access May 2, 2022
50 https://github.com/openssl/openssl, last access May 2, 2022
51 https://msol.io/blog/tech/create-a-self-signed-ecc-certificate/, last access May 2, 2022

Pi devices using the mbedtls[52] library. Executed commands duration is used to quantify computational requirements for the given operation. Additionally, created output, i.e., generated certificates or encrypted secrets, are measured to indicate memory requirements. Memory requirements to manage the list of revoked certificates are expressed through a single revocation entry with 100 Bytes, estimating the storage required to store the certificate's serial number and revocation timestamp.

However, in case of an inability to execute a cryptographic operation on the NodeMCU ESP32, input measurements are collected using the virtual environment for emulating resource-constrained devices. To simulate a weaker CPU, input measurements are collected using the command:

taskset –cpu-list 1 <command>,

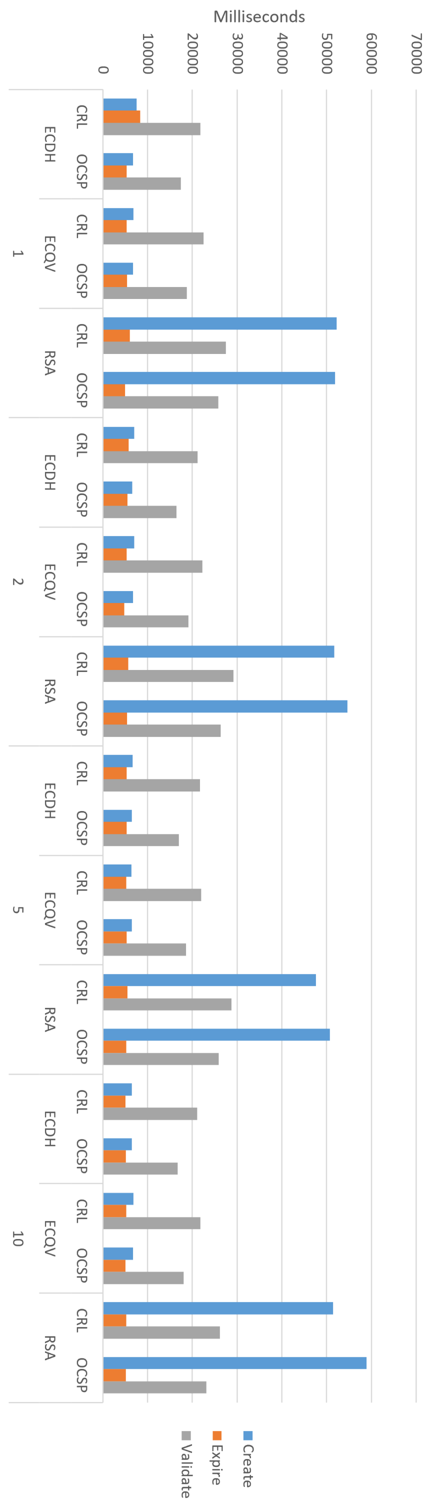
allowing execution of a command (e.g., asymmetric or symmetric data decryption) on one CPU thread.

Lastly, IoT network configuration is defined through its topology - the number of the Root, Intermediate, and Leaf Nodes and the used network protocols. To simulate wireless Smart Home devices, the simulations have been configured to use 2.4 GHz WiFi 802.11b [Sal+16a]. Data propagation losses between devices are using the fixed loss model with -70 dBm Received Signal Strength Indication, which corresponds to the medium quality links[53] between static, non-mobile Smart Home devices. As for the network size, to simulate the Smart Home scenario, 50 Leaf Nodes are configured. To evaluate the implication of Intermediate Node amount onto PKI efficiency, simulations were executed with various Leaf to Intermediate Nodes ratio. For this purpose, IoT network with 1, 2, 5, or 10 Intermediate Nodes have been chosen. The number of Root Nodes is fixed to one in all simulations.

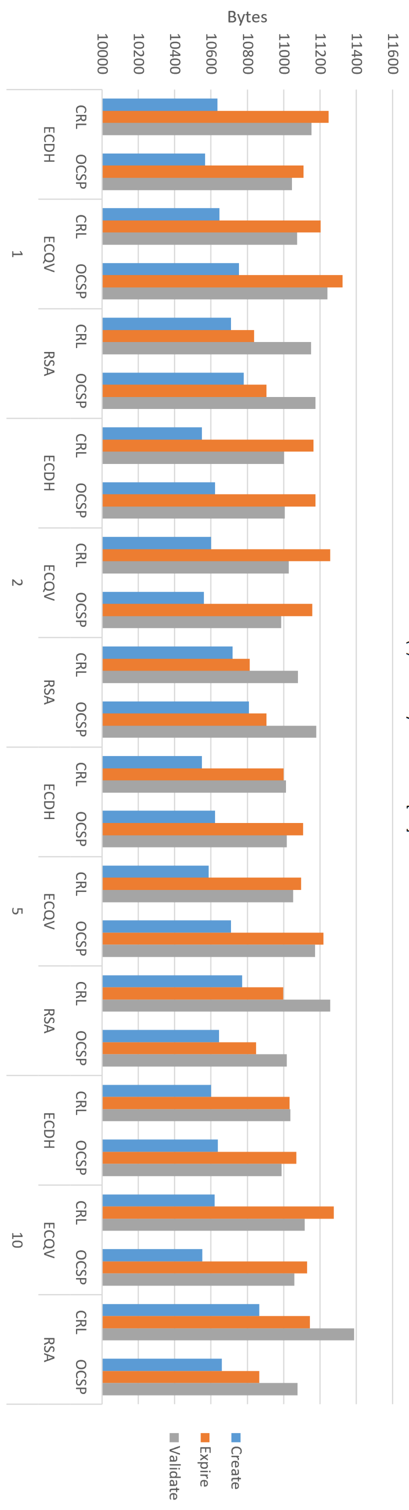
In order to simulate the above-mentioned scenarios, the following configurations are defined in Listings B.5, B.10, and B.15 for RSA, EC with DH (ECDH), and EC with QV (ECQV), respectively. Aggregated results concerning durations and memory consumption for *Create*, *Expire*, and *Validate* lifecycles are presented in Figure 5.3. Results are grouped from the bottom to the top on the x-axis by (i) Intermediate Nodes number, (ii) scenario, (iii) certificate revocation schema.

52 https://tls.mbed.org/, last access May 2, 2022

53 https://support.simplisafe.com/hc/en-us/articles/360035742191-What-is-WiFi-Strength-and-RSSI-, last access May 2, 2022



(a) PKI Lifecycle Duration [ms]



(b) PKI Lifecycle Memory Consumption [Bytes]

Figure 5.3: PKI simulation results

The first finding concerning presented results indicates that OCSP outperforms the CRL certificate revocation scheme for each simulation scenario, both in terms of the average lifecycle duration and memory consumption. Since OCSP queries are hosted by the CA node, Intermediate or Root Node, despite introducing additional network communication between Leaf Node and CA node, the computational capabilities of those nodes outperform the lookup in the locally stored CRLs in Leaf Nodes.

The second finding focuses on choosing the most efficient encryption and KMP schema. Figure 5.3 shows a significant duration increase for the RSA encryption for the *Create* and *Validate* lifecycles, which is caused by the RSA's computational complexity and required key-length to achieve the same security level as EC. For that reason, closer analysis between ECDH and ECQV is performed.

By comparing the presented memory consumption between ECDH and ECQV for each lifecycle with OCSP, it can be concluded that ECDH requires slightly less memory for the *Expire* and *Validate* lifecycles, while it is quite similar for the *Create* lifecycle. For example, with one Intermediate Node, ECDH requires 11109 Bytes and ECQV 11322 Bytes on average for the *Expire* lifecycle, which corresponds to a memory consumption difference of 1.88%. Similar values are valid for multiple Intermediate Node scenarios: (a) 11174 Bytes for ECDH and 11157 Bytes for ECQV with two Intermediate Nodes, making ECQV consumer 0.15% less memory, (b) 11106 Bytes for ECDH and 11219 Bytes for ECQV with five Intermediate Nodes, resulting in 1% less memory consumption for ECDH, and (c) 11070 Bytes for ECDH and 11128 Bytes for ECQV with 10 Intermediate Nodes, indicating that ECDH requires 0.52% less memory. These values give a minor advantage to the ECDH but cannot be the determining factor for the final decision.

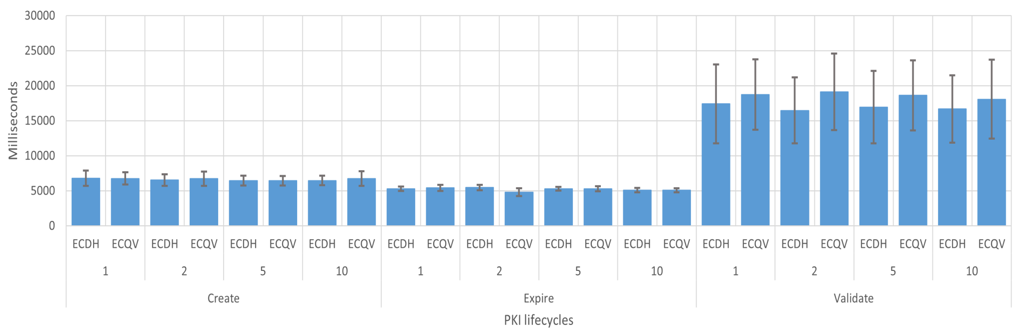


Figure 5.4: Lifecycle duration comparison between ECDH and ECQV

Further evaluations presented in Figure 5.4 provide an analysis of the average and standard deviation values for the duration of the lifecycles. This analysis considers ECDH and ECQV using OCSP and different Intermediate Nodes numbers. As the compared values for *Create* and *Expire* lifecycles are similar between ECDH and ECQV, the *Validate* lifecycle has been examined more closely.

Compared to the other lifecycles, the *Validate* lifecycle values indicate its significantly longer duration and a bigger standard deviation. The lifecycle's complexity mostly causes these increases since the certificate validation requires communication between multiple network nodes (cf. Appendix B). Due to that, the validation in a node is more affected by the factors like network congestion, transmission delay, and communicating node processing latency, causing a higher deviation from the average value. Concerning standard deviation, both ECDH and ECQV have similar values, ranging from 1 to 10 percent in comparison, indicating that both options are similarly affected by the factors mentioned above.

With regard to the average lifecycle duration values, ECDH outperforms ECQV for any Intermediate Node's number:

1. 1336 ms or 7.13% for one Intermediate Node,
2. 2660 ms or 13.89% for two Intermediate Nodes,
3. 1673 ms or 8.98% for five Intermediate Nodes, and
4. 1377 ms or 7.61% for 10 Intermediate Nodes.

This indicates that the Implicit certificate's lesser size does not bring significant benefits, despite reducing the bandwidth consumption required for the certificate transmission between nodes. For that reason, it has been decided that the ECDH is the best option for implementing the certificate management.

Furthermore, these results mean that Leaf Nodes' (Things') computational capabilities represent a bottleneck in the KMPs, which should be reduced through Intermediate Nodes (FNs). Therefore, an additional analysis is done to evaluate if the higher number of Intermediate Nodes can reduce average lifecycle duration. As presented in Figure 5.4, more Intermediate Nodes can slightly reduce the average lifecycle durations. For example, comparing the scenario containing two Intermediate Nodes with the scenario containing one Intermediate Node, the lifecycle duration for ECDH is reduced: (a) from 6784 ms with one to 6554 ms with two Intermediate Nodes, which is 230 ms or 3.55% decrease for *Create* lifecycle or (b) from 17409 ms with one to 16478 ms with two Intermediate Nodes, resulting in 931.11 ms or 5.65% decrease for *Validate* lifecycle. Similar results remain for scenarios with five and 10 Intermediate Nodes. For that reason, it can be concluded that adding additional Intermediate Nodes would not significantly speed up KMPs since Leaf Nodes remain the critical bottleneck.

Results and analysis show that each computations reduction on the Leaf Nodes brings performance benefits to the overall PKI: (1) using CA-based OCSP queries instead of locally managed CRLs, (2) using shorter encryption keys, and (3) avoiding generating certificates for *Validation* lifecycle using ECQV. Due to that, ECDH with OCSP has been chosen as the best approach for the trust management implementation. Further analysis has shown that adding additional CA nodes do not significantly improve overall performance. For that reason, more lightweight approaches have to be introduced to bring not just reduced latency in the IoT network but also battery- and memory-saving benefits for Things.

5.3 Trustworthy Networking

Establishing TN services for FC services deployment and securing IoT devices and users represents this dissertation's significant contribution. This involves the implementation of the complete trust management lifecycle with initialization, validation, and revocation (cf. Section 4.4). To evaluate the feasibility of the implemented solution concerning its scalability and introduced latency, the performance of the trust management-related operations have been measured. The first performance evaluation presented in Section 5.3.1 analyses trust and certificate management operations described in Sections 4.4.1 and 4.4.2, respectively. The second performance evaluation documented in Section 5.3.2 examines achieving best-effort E2E security for IoT devices, based on the security profiles and implemented in Section 4.4.3.

5.3.1 *Certificate & Trust Management*

Enabling TN services in the COSYLab requires FC services to initialize their trust management operations. Initialization involves security credentials creation and exchange in order to mutually authenticate COSYLab services (cf. Section 3.2.2) and issue digital certificates presented in Section 4.4.1. The used digital certificates are based on ECC with the curve named *"secp256r1"* [NJP18] and ECDSA public key pair generating algorithm [Por13]. The conducted performance evaluation examines TN services initialization routines and their applicability on the different devices serving as FNs described in Section 5.1.

Table 5.5: Latencies for trust bootstrap stages between FTA and TNTA

Operation	Raspberry Pi 4	Raspberry Pi 3	Raspberry Pi Zero
Generate identity	79 ms	143 ms	7112 ms
Generate and store public key pair	339 ms	1500 ms	938002 ms
Register credentials at TNTA	1967 ms	3112 ms	395396 ms
Decrypt CSR token and create CSR	1540 ms	2447 ms	5988371 ms
Obtain certificate from TNTA	171 ms	183 ms	15667 ms
Store private key and certificate	355 ms	225 ms	30631 ms
Overall duration	4451 ms	7610 ms	7375179 ms

The initial step for provisioning FC-based TN services is for FTA to join CTFTP and retrieve its digital certificate, resulting in enabled TN services in local networks without dependency on Cloud's availability. The stages occurring in the process of FTA joining CTFTP are presented in Figure 3.6. The duration of each stage is measured and analyzed concerning the TN services applicability on the FN devices, leading to the results presented in Table 5.5.

Based on the presented results Raspberry Pi 4 and Raspberry Pi 3 devices can perform required operations in a reasonable amount of time, i.e. a couple of seconds. Generating identities based on the naming structure documented in Section 4.4.1 lasts a neglectable amount of time. The duration of the stages involving cryptographic operations, i.e., generating public key pair, hashing credentials for TNTA registration, decrypting CSR token, and digitally signing CSR, is longer due to the computational complexity of the involved operations. Nevertheless, the overall duration of a couple of seconds for all stages proves that Raspberry Pi 4 and Raspberry Pi 3 devices can be used as FNs for FTA service.

In contrast to that, Raspberry Pi Zero takes a lot more time to finish all stages. The computational complexity of the involved cryptographic operation combined with the required communication with TNTA extends the overall duration to approx. 2 hours. Considering that all the stages take place during the first startup of COSYLab services, Raspberry Pi Zero would introduce delay in IoT system installation that hardly can be acceptable in IoT systems.

Once FTA has joined the CTFTP, it can host TN services for FTTTP entities: FC-services, Things, and users. To evaluate the feasibility of deploying TN services in the FTTTP, the trust bootstrap procedure between FC-services and FTA described in Figure 3.7 is observed. Within the scenario described in Section 4.1, FTP has been selected as an example FC-service, and it joins FTTTP by obtaining its digital certificate from FTA.

Measured durations of the stages required for FTP to join FTTTP are presented in Table 5.6. Compared to the results presented in Table 5.5, Raspberry Pi 4 and Raspberry Pi 3 devices perform similarly, leading to the nearly equal durations required for services to join CTFTP and FTTTP. Furthermore, Raspberry Pi Zero performs much better in the FTTTP joining scenario. Better performance is caused by fewer cryptographic operations included in evaluated stages, i.e., credentials are not hashed for registration, or encrypted CSR token is not exchanged in FTTTP procedures. This results in a completely missing stage *Register credentials* and reduced duration for *"Create CSR"* stage from 5988371 ms to 29270 ms. Additionally, FC services dominantly communicate in the local network, minimizing latency involved in communication with the remote TNTA service.

Table 5.6: Latencies for trust bootstrap stages between FTP and FTA

Operation	Raspberry Pi 4	Raspberry Pi 3	Raspberry Pi Zero
Generate identity	100 ms	140 ms	7509 ms
Generate and store public key pair	1891 ms	2986 ms	395204 ms
Obtain FTA certificate	1021 ms	2087 ms	99194 ms
Validate FTA certificate	846 ms	954 ms	96354 ms
Create CSR	145 ms	217 ms	29270 ms
Obtain certificate from TNTA	70 ms	89 ms	1939 ms
Store private key and certificate	489 ms	995 ms	385043 ms
Overall duration	4562 ms	7468 ms	1014513 ms

The above mentioned factors lead to the overall duration for FTP on Raspberry Pi Zero FTTTP joining procedure of approximately 17 minutes, which can be tolerable startup latency for FC services depending on their criticality. For example, FACA is required for a direct user interaction during the IoT system's setup, making it a bad candidate to be deployed on Raspberry Pi Zeros. In contrast, FTP and C-A agents are background services. They do not affect user interaction during IoT system setup, making them good candidates to be deployed on the less powerful Raspberry Pi Zero FNs. Raspberry Pi 4 and Raspberry Pi 3 devices have shown much better performances than Raspberry Pi Zero. A couple of seconds latency for trust bootstrapping for both devices makes them suitable for hosting FTA and FTP services, without a higher impact on COSYLab's performances.

5.3.2 *Security Profiles*

Besides identity and certificate management, the implemented services offer E2E security services for Things through the FTP component (cf. Section 4.4.3). Through the FTP services, Things are recognized in the implemented trust management solution and can mutually authenticate with other IoT entities. E2E security services rely on security profiles, which adapt to the Things' computational capabilities. The FTP component incorporates security profiles and adjusts Things credentials and message exchanges accordingly. Computational load on FTP and its scalability are affected by the application of security profiles and the number of connected Things. For those reasons, performances of the FTP service for signing messages produced by Things are evaluated.

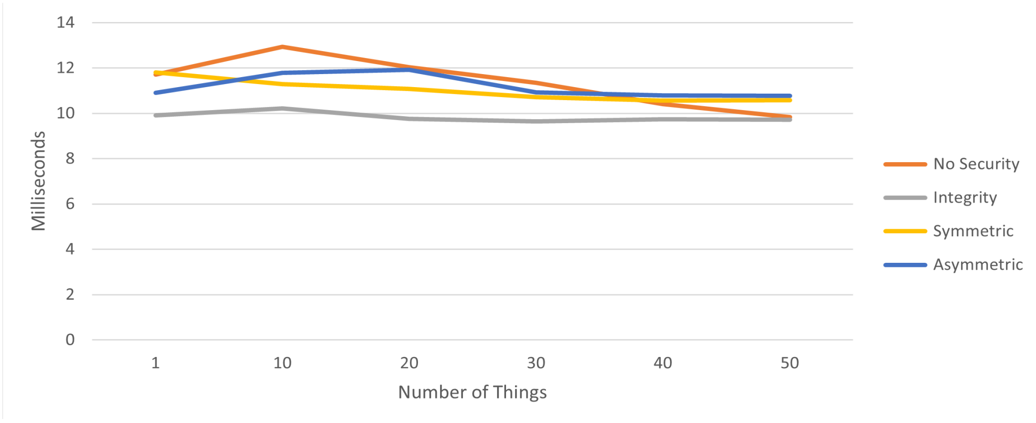


Figure 5.5: FTP processing latency Raspberry Pi 4

Performance evaluation focuses on measuring E2E latency between Things and FTP service. In the scenarios, a various number of Things send message signing requests to FTP every 10 seconds. The FTP component is deployed on the same Raspberry Pi devices used in the evaluation described in Section 5.3.1. The number of connected Things using FTP service resembles the number of IoT devices in a Smart Home (cf. Section 4.1) and varies in tests with values: 1, 10, 20, 30, 40, and 50, and the following security profiles:

1. The *No security* profile implies plain message exchange, involving no cryptographic operations on the FTP side once a message from Thing is received;
2. The *Integrity* profile involves checking if the received message hash is correct based on the SHA-256 hashing algorithm;
3. The *Symmetric, secure keys* profile involves decrypting the received message using AES encryption with 256 bits key length;
4. The *Asymmetric, secure raw public key* profile involves decrypting the received message using EC encryption with 256 bits key length, based on certificates presented in Section 4.4.1.

The first analysis focuses on the scalability of FTP services concerning (1) the number of connected Things and (2) applied security protocols. To achieve this, processing latency in the FTP message signing service on each Raspberry Pi device has been measured. The analysis results containing the average processing latencies are presented in (i) Figure 5.5 for Raspberry Pi 4, (ii) Figure 5.6 for Raspberry Pi 3, and (iii) Figure 5.7 for Raspberry Pi Zero device.

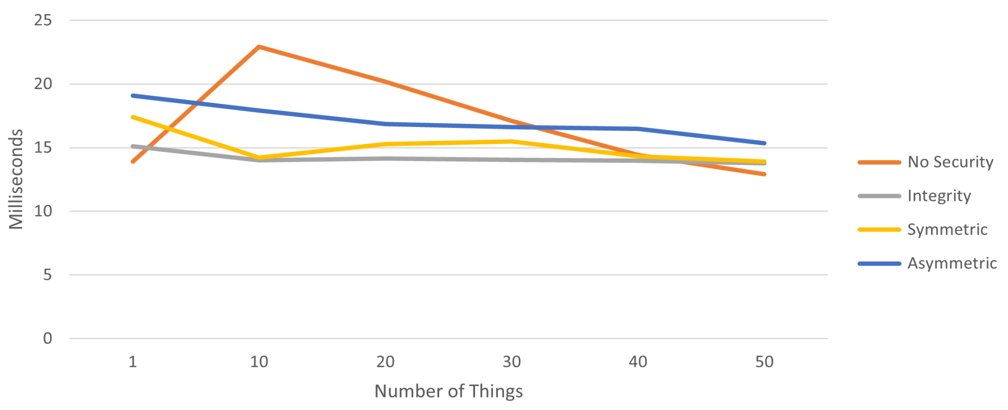


Figure 5.6: FTP processing latency Raspberry Pi 3

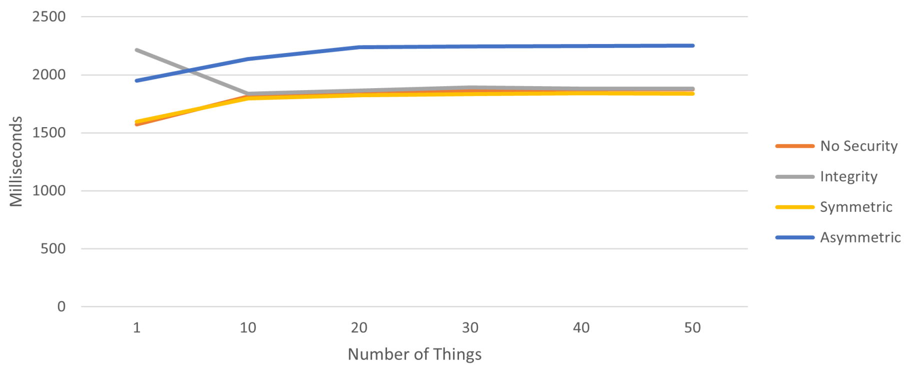


Figure 5.7: FTP processing latency Raspberry Pi Zero

The presented figures show that the connected Things number has a small impact on FTP processing latency. On Raspberry Pi 4 and Raspberry Pi 3 devices, no steady processing latency growth correlating with the connected Things increase is detected. Raspberry Pi Zero shows a minor processing latency increase correlating with the connected Things number for most security profiles. This biggest increases are present between scenarios with 1 and 50 Things for *No Security* - 302 ms (19.2 %) or 6.04 ms (0.38 %) per Thing and *Asymmetric* - 301 ms (15.45 %) or 6.03 ms (0.31 %) per Thing. These increases are negligible for a Smart Home, proving that the FTP service scales with the increase of the connected Things.

The second analyzed aspect represents the impact of the security profile on a processing latency. To that regard, all devices have shown the negligible processing time difference for *No Security*, *Integrity*, and *Symmetric* security profiles. The only security profile that implied increase in processing latency is *Asymmetric* since it involves computationally demanding operations with the public-key cryptography. For scenarios with 50 connected Things, the *Asymmetric* profile compared to the *No Security* profile introduces a processing latency increase: (1) 0.93 ms, that is 9.5% for Raspberry Pi 4, (2) 2.45 ms, that is 19% for Raspberry Pi 3, and (3) 376 ms, that is 20.03% for Raspberry Pi Zero device.

Table 5.7: Average processing latency with 50 Things per security profile

Device	No Security	Integrity	Symmetric	Asymmetric
Raspberry Pi 4	9.84 ms	9.71 ms	10.58 ms	10.77 ms
Raspberry Pi 3	12.91 ms	13.78 ms	13.89 ms	15.36 ms
Raspberry Pi Zero	1875.87 ms	1881.2 ms	1837.9 ms	2251.76 ms

The second analysis focuses on the FTP service responsiveness concerning the FN devices' computational capabilities. In the scenario with 50 connected Things, the FTP service average processing latencies for every security profile have been measured. The results are summarized in Table 5.7. Raspberry Pi 4 and Raspberry Pi 3 devices show outstanding performance since their processing latency remains under 16 ms for each profile. These performances satisfy most latency-critical IoT applications [Sch+17]. Raspberry Pi Zero processing latency revolves around 2 seconds which makes it unsuitable for latency-critical IoT applications. Still, deploying FTP services on Raspberry Pi Zero devices can increase IoT system's robustness and fail-safety in non-latency-critical IoT application domains.

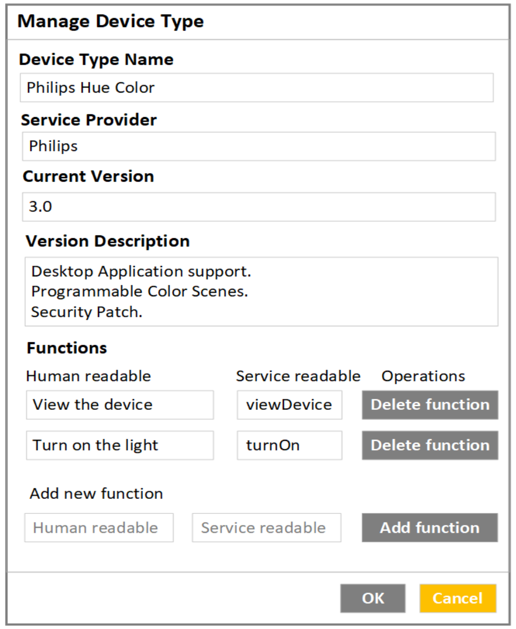
5.4 Access Control

One of the the main goals of this dissertation is to provide the trustworthy distribution of AC in FC-based IoT systems to increase the overall robustness and performance of AC in IoT. To validate this, the developed AC components - ACAM and FACA - were deployed and tested in the Smart Home system described in Section 4.1. The performed evaluation is twofold: (1) A proof of operability for maintaining a consistent state between access policies, and (2) a performance evaluation measuring solutions' impact on the authorization latency.

5.4.1 *Synchronization of the Security Policies Configuration - Proof of Operability*

Based on the requirements for FC-based IoT AC systems outlined in Section 3.3, the implemented AC distribution approach involves FN-hosted security policies management (cf. Section 4.6.2) and synchronization of the policy configuration between Cloud Server and FN presented in Section 4.6.3. To prove the feasibility of this approach and proper AC provisioning, web applications present in COSYLab are used - CSWA for Cloud Server and Fog Controller for FN.

Considering a Things' management in COSYLab, the following features were identified as occurring in a systems' lifecycle and as vital for the systems' operability, with special focus on AC configuration distribution: (F1) Initial AC setup, (F2) Definition of access rights, (F3) Update of AC configuration, and (F4) Access rights validation.



Manage Device Type

Device Type Name
Philips Hue Color

Service Provider
Philips

Current Version
3.0

Version Description
Desktop Application support.
Programmable Color Scenes.
Security Patch.

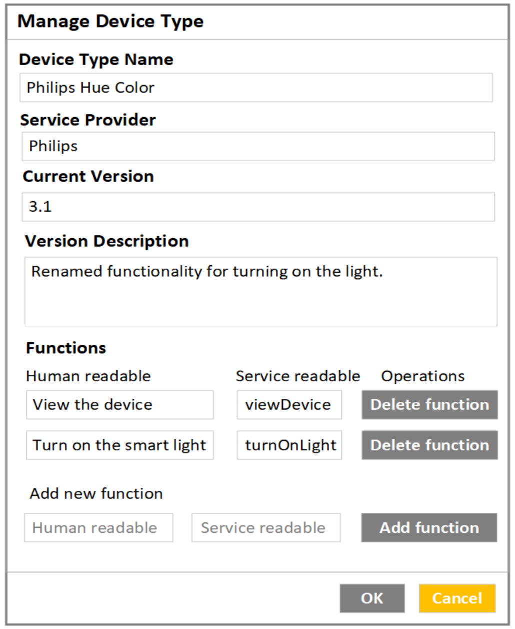
Functions

Human readable	Service readable	Operations
View the device	viewDevice	Delete function
Turn on the light	turnOn	Delete function

Add new function

Human readable	Service readable	Add function

(a) Initial Device Type setup



Manage Device Type

Device Type Name
Philips Hue Color

Service Provider
Philips

Current Version
3.1

Version Description
Renamed functionality for turning on the light.

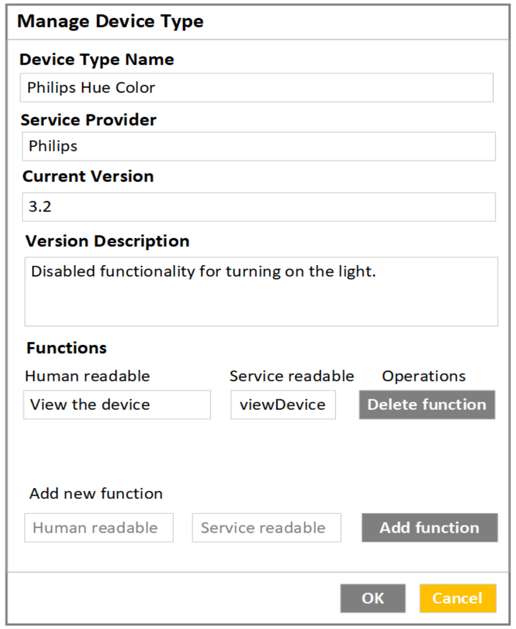
Functions

Human readable	Service readable	Operations
View the device	viewDevice	Delete function
Turn on the smart light	turnOnLight	Delete function

Add new function

Human readable	Service readable	Add function

(b) Device Type update – renaming the device's functionality



Manage Device Type

Device Type Name
Philips Hue Color

Service Provider
Philips

Current Version
3.2

Version Description
Disabled functionality for turning on the light.

Functions

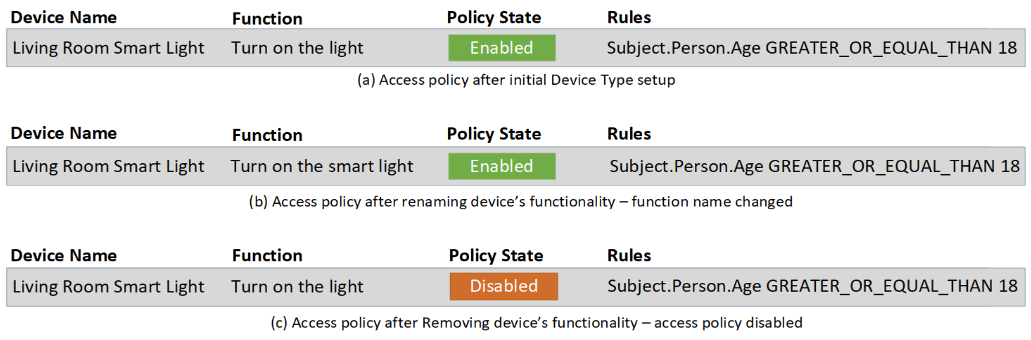
Human readable	Service readable	Operations
View the device	viewDevice	Delete function

Add new function

Human readable	Service readable	Add function

(c) Device Type update – deleting the device's functionality

Figure 5.8: ACAM device type configuration



Device Name	Function	Policy State	Rules
Living Room Smart Light	Turn on the light	Enabled	Subject.Person.Age GREATER_OR_EQUAL_THAN 18

(a) Access policy after initial Device Type setup

Device Name	Function	Policy State	Rules
Living Room Smart Light	Turn on the smart light	Enabled	Subject.Person.Age GREATER_OR_EQUAL_THAN 18

(b) Access policy after renaming device's functionality – function name changed

Device Name	Function	Policy State	Rules
Living Room Smart Light	Turn on the light	Disabled	Subject.Person.Age GREATER_OR_EQUAL_THAN 18

(c) Access policy after Removing device's functionality – access policy disabled

Figure 5.9: FACA device configuration update

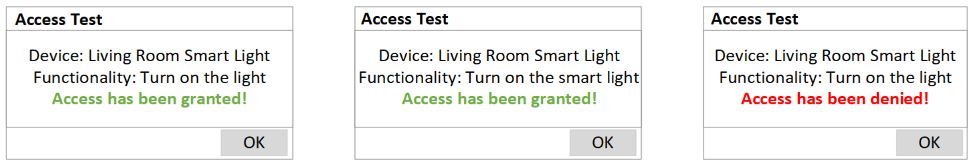
In order to show that these functionalities F1-F4 are supported by ACAM and FACA, a Smart Light device[54] has been connected to the Fog controller. An AC configuration the through device type versions depicted in Figure 5.8 has been defined in ACAM, causing adaptations of the access policies in FACA for limiting access outlined in Figure 5.9. Due to these adaptations, the access rights validations for a particular functionality, i.e. turning on the Living Room Smart Light, have been affected as documented in Figure 5.10.

The first step is presented as the scenario *(a)* on the graphics mentioned above. This scenario describes the F1 functionality, that is the device type setup in ACAM, enabling the access policy definition (F2) in FACA (cf. Figure 5.9a). Once the access policy is defined, enabling all persons older than 18 the possibility to turn on the smart light, access to the IoT service is granted (cf. Figure 5.10a).

The next scenario is the AC configuration update - renaming IoT service (cf. Figure 5.8b) to the new name *"Turn on the smart light"*. Once the FACA's AC configuration is synchronized with the ACAM's update, the access policy remains enabled with the same access rules (cf. Figure 5.9b), enabling further access to the renamed IoT service (cf. Figure 5.10b).

The final step is to update the initial AC configuration by disabling the functionality mentioned above once the AC configuration is distributed from ACAM to FACA, as presented in Figure 5.8c. From that moment on, access to the disabled functionality is denied for all users, independently on the defined access rules in FACA (cf. Figure 5.10c), since the functionality is no longer considered as supported and all attached access policies are disabled by FACA (cf. Figure 5.9c).

54 https://www2.meethue.com/de-at, last access May 2, 2022


Access Test

Device: Living Room Smart Light
Functionality: Turn on the light
Access has been granted!

OK

(a) Initial Device Type setup access policy validation

Access Test

Device: Living Room Smart Light
Functionality: Turn on the smart light
Access has been granted!

OK

(b) Renamed functionality access policy validation

Access Test

Device: Living Room Smart Light
Functionality: Turn on the light
Access has been denied!

OK

(c) Deleted functionality access policy validation

Figure 5.10: FACA access policies validation

5.4.2 *Authorization Performance Analysis*

IoT users require an immediate system-independent management of settings and security policies updates. Thus, long delays due to processing are unacceptable in general and should be dependent of the size of the network and the system's security configuration. In order to evaluate this, delay measurements were performed for different network sizes to prove that the impact of access policy validation in the local network is in an acceptable range. For that purpose, the COSYLab Fog Controller and FACA modules are installed on a Raspberry Pi 4 FN device (cf. Section 4.1). In order to provide better insights into access policy validation times, average latencies and latencies distribution are analyzed, examining the occurring network, processing, and E2E latencies and covering the full impact of the authorization logic on the IoT system's operability.

First, the impact of the used application layer protocol on the network latency between Fog Controller and FACA is analyzed by comparing AMQP and HTTP protocols. For this purpose, an evaluation scenario with five IoT users, each one concurrently requesting 10 access rights validation, is defined. The targeted access policy consists of five different access rules, evaluating various user's attribute values. Through that, extensive load on FACA regarding validation requests processing and queueing, is created. As presented in the results in Figure 5.11, the HTTP protocol allows a higher level of concurrent requests execution, which puts more load on FACA and increases overall processing latency. However, increased processing latency is negligible compared to the network and E2E latency introduced by AMQP due to the serialization of the request on the RabbitMQ broker. Therefore, the average E2E latency comparison clearly speaks in favor of HTTP, leading to the conclusion that the HTTP protocol should be used for communication between Fog Controller and FACA in case of a high load concerning the access policy validation.

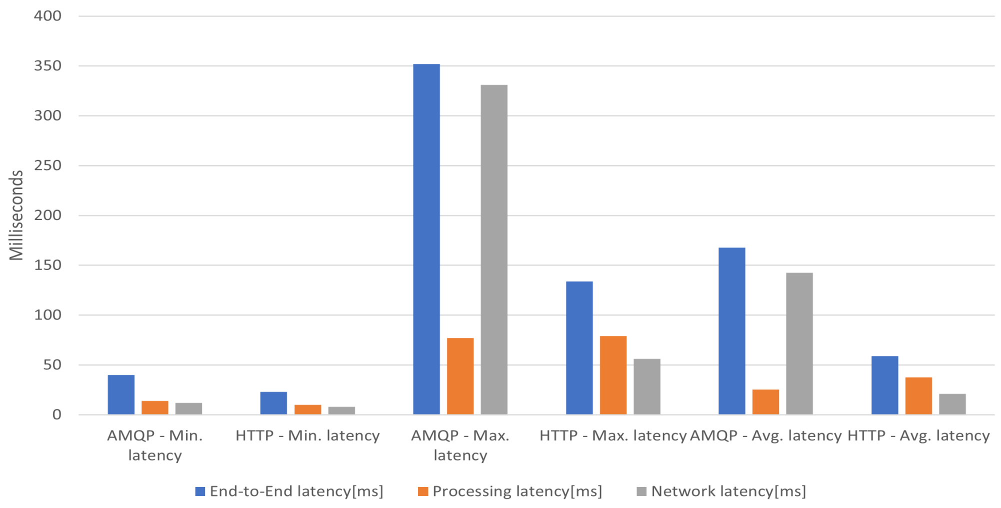


Figure 5.11: FC components network protocols comparison

Table 5.8: Access policy performance evaluation cases

Aspect	Users number	Access rules per policy	Requests per user
Policy complexity	5	1, 5, 10, 15, 20	50
User number	1, 2, 3, 4, 5	5	50
Concurrent users scalability	1, 20, 40, 60, 80, 100	5	10

As the FACA is installed on a FN with limited computational power and equipped with an XACML-based access policy validation engine, the evaluation analyzed how further factors influence the access policy validation delay:

1. Access policies complexity, that is, the number of access rules defined in a single access policy;
2. The number of users using the system simultaneously;
3. FACA's scalability concerning the number of concurrent access policy validation requests.

An overview of the used values for these factors in different performance evaluation aspects is provided in Table 5.8. *Policy complexity, user number, and requests per user* performance evaluation aspects involve values that are defined using the estimation of the Smart Home environment described in Section 4.1, with each

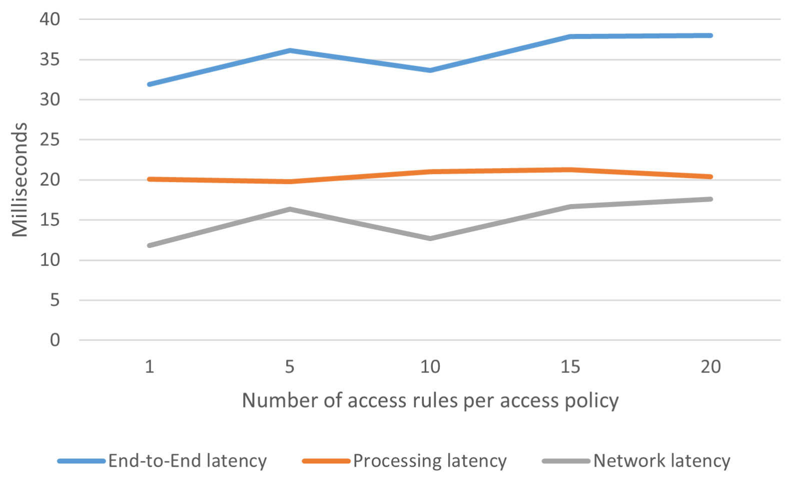


Figure 5.12: FACA access policies complexity average latency

member issuing a batch of validation requests (up to 100). Each validation request targets an access policy of different complexity, involving verification of up to 20 access rules, making use of ABAC for managing fine-grained access policies. The *Concurrent users scalability* performance evaluation aspect represents a stress test, creating extensive processing load for the FACA, therefore analysing FACA's applicability in IoT application domains other than Smart Home (e.g., Smart Building or Smart Office) since the foreseen number of users in this evaluation aspect is up to 100.

Since FACA allows the definition of fine-grained access policies and enables the analysis of multiple factors during the validation of the users' access rights, the effect of access policy complexity on the access rights validation delay has been analyzed. In order to achieve that, access policies with 1, 5, 10, 15, and 20 access rules have been defined and used during the evaluation. Access rules in these policies represent one logical condition, which is to be validated for its correctness.

As presented in Figure 5.12, a slight impact of the number of access rules on the validation time of access policies can be noticed. Namely, average E2E latency increases between scenarios with 1 and 5 access rules in the access policy from 31.91 ms to 36.1 ms, which represents 13% increase. Afterward, a minor E2E latency increase from 36.1 ms to 37.99 ms, that is 5%, between scenarios with 5 and 20 access rules is noticeable. However, the E2E latency increase is mainly impacted by the network latency since the processing latency in FACA remains almost constant in all test scenarios with a variable access policy complexity.

Table 5.9: FACA access policies complexity latency distribution in %

Latency limit [ms]	1 Access Rule	5 Access Rules	10 Access Rules	15 Access Rules	20 Access Rules
350.00	0.0	0.0	0.0	0.0	0.0
315.00	0.0	2.0	0.0	0.0	0.0
280.00	0.0	0.0	0.0	2.0	2.0
245.00	0.0	0.0	0.0	0.0	0.0
210.00	0.0	0.0	0.0	0.0	0.0
175.00	0.0	0.0	0.0	0.0	0.0
140.00	0.0	0.0	0.0	0.0	0.0
105.00	0.40	1.60	4.0	2.0	0.80
70.00	35.60	28.4	36.80	33.60	37.20
35.00	64.0	68.0	59.20	62.40	60.0

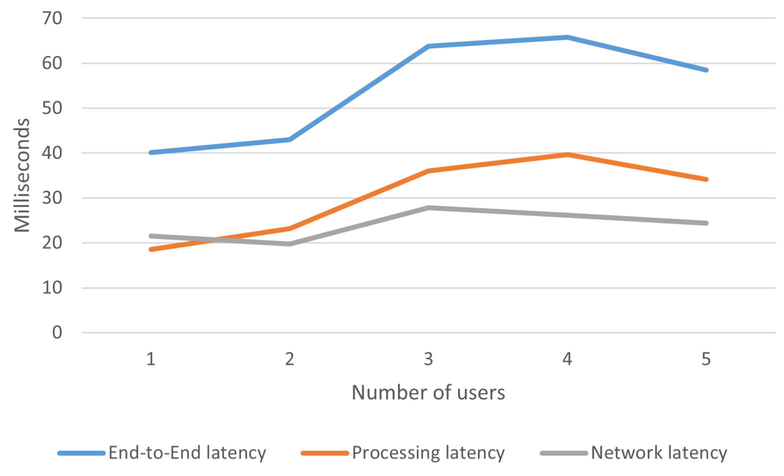


Figure 5.13: FACA user number average latency

E2E latencies distribution concerning the access policy complexity is presented in the heatmap in Table 5.9. The heatmap presents the percentage of authorization requests that were executed below a certain latency limit. The heatmap results also indicate the E2E latency growth with the increase of access policy complexity, since the requests number steadily shift from a 35ms to a 70ms latency limit as the access policy complexity increases. However, in each test scenario, a minimum of 96% of requests are executed within 70ms. Hence, no significant correlation between the access policy complexity and the impact on the access policy validation latency has been found, implying that the access policy complexity does not jeopardize FACA's scalability.

Since the developed solution is used in multi-user environments, FACA's scalability concerning the number of concurrent users in the IoT system has been evaluated. For this purpose, to represent a most common European household, a variable number of simultaneous Smart Home users (1 - 5) has been emulated in COSYLab, each of them triggering the validation of 50 access policies, with five access rules each. The results presented in Figure 5.13 show that an increase in the number of users concurrently issuing the access policy validation requests increases overall IoT system's latency. The average E2E latency increased by 46%, that is from 40.08 ms in scenario with one to 58 ms in scenario with five users, which is mainly impacted by the processing latency in FACA, since FACA processing latency increased by 84%, from 18.52 ms to 34 ms.

However, the E2E latencies distribution presented in Table 5.10 shows that an E2E latency increase is caused mainly by several requests that require a lot more time to execute than most requests. This occurs more frequently as the load on FACA increases with the user number (3, 4, 5), as this introduces a higher probability for creating a processing bottleneck in FACA. Despite of that, 92% of requests in each scenario are executed within 105 ms, indicating that the user number does not dramatically decrease overall FACA's performance. Still, the user number has a more significant impact on E2E latency than the access policy complexity.

Examining a detected scalability bottleneck in depth requires performing a scalability test, aiming at a IoT environment with a higher number of users - Smart Building. In this scenario, up to 100 users are simulated, each sending ten validation requests, creating a higher load on FACA through concurrent validation requests. Results in Figure 5.14 show a direct correlation between the user number and latency increase and the leading cause of the processing bottleneck. Namely, E2E and processing latency growth are approximately linear for each E2E latency. The most considerable average E2E latency increase is between 1 and 20 users - from 32 ms to 332 ms, resulting in 6.3 ms or 35% increase per added user, while the processing latency growth is from 7.6 ms to 97.96 ms, that is, 59.45% per added user. Further scenarios with additional 20 users each show constant average E2E latency increases for each added user: (a) 143.75 ms for 20 to 342.43 ms for 40 users (9.93% per user), (b) 342.43 ms for 40 to 396.54 ms for 60 users (2.7% per user), (c) 396.54 ms for 60 to 473.61 ms for 80 users (3.85% per user), and (d) 473.61 ms for 80 to 507.52 ms for 100 users (1.69% per user). This indicates FACA's lacking scalability with a higher number of concurrent users.

Table 5.10: FACA user number latency distribution in %

Latency limit [ms]	1 User	2 Users	3 Users	4 Users	5 Users
350.00	0.0	0.0	0.0	0.0	0.0
315.00	0.0	0.0	2.0	0.0	2.0
280.00	0.0	0.0	0.0	0.5	0.0
245.00	0.0	0.0	0.0	0.0	0.0
210.00	0.0	0.0	0.0	0.0	0.0
175.00	2.0	0.0	0.0	1.0	0.0
140.00	0.0	0.0	1.33	6.5	0.0
105.00	2.0	7.0	24.0	31.0	1.60
70.00	54.00	61.0	67.33	50.50	28.40
35.00	42.0	32.0	5.33	10.50	68.0

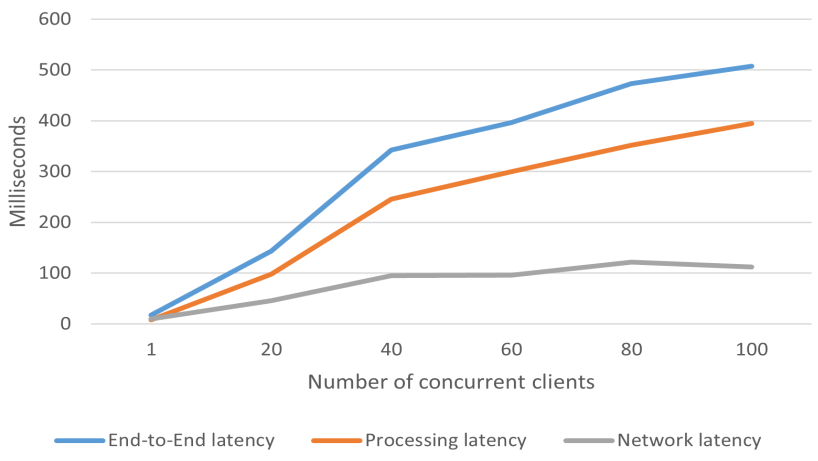


Figure 5.14: FACA's scalability verification - average latency

Similar observations can be derived from the heatmap presented in Table 5.11. Namely, every 20 new users significantly increase E2E latency distribution, shifting more than 90% of requests latency in each scenario from (a) under 160ms for one user to (b) under 640ms for 20 and 40 users and (c) under 960ms for 60, 80 and 100 users. These results indicate that the developed FACA component scales with performances in the Smart Home environment, but its application in IoT application domains with a higher user number requires further distribution of the processing load between multiple FACA components. This would reduce the processing load on a single FACA container and the caused access policy validation latency.

Table 5.11: FACA's scalability verification - latency distribution in %

Latency limit [ms]	1 User	20 Users	40 Users	60 Users	80 Users	100 Users
1600.00	0.0	0.0	0.0	0.0	0.25	0.10
1440.00	0.0	0.0	0.0	0.0	0.38	1.30
1280.00	0.0	0.0	0.75	0.0	1.5	1.7
1120.00	0.0	0.0	1.25	0.17	3.0	3.8
960.00	0.0	0.0	2.75	1.5	6.0	8.0
800.00	0.0	0.0	5.25	2.33	12.5	14.6
640.00	0.0	0.0	9.5	11.83	20.88	19.7
480.00	0.0	1.0	29.0	24.5	25.625	23.5
320.00	0.0	36.0	30.75	38.83	19.66	17.4
160.00	100.0	63.0	20.75	20.83	10.38	9.9

5.5 Context-Aware Access Control

Context information integration into AC mechanisms allows the definition of security policies that adapt to the IoT environment's state. However, introduced integration increases additional processing requirements compared to the AC solution, which is evaluated in Section 5.4. The developed C-A AC solution was deployed and verified using COSYLab and the scenario described in Section 4.1. Thereby, the C-A AC solution's verification is threefold: (1) the proof of operability through the adaptation of the security policies to the change in IoT environment, (2) the performance evaluation of C-A-based authorization (3) latency overhead comparison between C-A AC and AC evaluation in Section 5.4.2.

5.5.1 *Adaptable Access Policies Proof of Operability*

Guided by the requirements for developing C-A solutions defined in Section 3.4, the implemented C-A AC solution enables integrating context information into security policies (cf. Section 4.7). To prove the solution's usability and operability, functional evaluation was conducted relying on the implemented Smart Home management system described in Section 5.4.1. Identified main building blocks for the functional evaluation are based on the stages in the communication between C-A agents, FACA, and the user. They are split as follows:

- (S1) C-A attribute registration by C-A agent;
- (S2) access policy definition by the user;
- (S3) C-A attribute value notification by C-A agent;
- (S4) access policy validation by FACA.

In the stage S1, the CCAA agent registers the C-A attribute through its configuration at FACA, presented in Listing 4.23. FACA afterward uses the C-A attribute configuration to present the access policy definition options to the user. A user chooses the registered C-A attribute *Context.Connectivity.ConnectedToCloud* for the defined access policy (S2). Along with the C-A attribute the user specifies that the access policy will be satisfied if CCAA is connected to the Cloud Server and has more than 70% confidence in that information (cf. Listing 5.1).

Once the C-A attribute is registered successfully, CCAA monitors its connection to the Cloud Server and notifies C-A attribute's value to FACA (S3). In the first step, presented in Listing 5.2, the CCAA's connection to the Cloud Server works without any interruptions, and therefore it notifies a working connection to FACA with 100% certainty. Once the C-A attribute value is notified, the user initiates the access policy validation using a notified value, and since it satisfies the predefined access rules, the user is granted an access to the requested IoT service, as shown in Figure 5.15a.

Enabling C-A authorization requires the more complex authorization logic presented in Listing 4.27, so that the context information is included into the access policy validation. To achieve that, attribute values are not collected just from a user's session token, but also from the database and directly from C-A agents. In order to evaluate this, processing, network, and E2E latencies in different C-A authorization scenarios are measured. The performance evaluation testbed reuses the setup described from Section 5.4.2 to create comparable results to the ones in Section 5.4.2 and measure overhead introduced by C-A overhead outlined in Section 5.5.3.

In the second step, the connection to the Cloud Server has been interrupted for two minutes. CCAA detects a missing connection and decreases the certainty in the given attribute value to 0%, indicating a complete absence of Cloud server's availability. Once the Cloud Server has recovered and connection has been reestablished, CCAA starts increasing the certainty levels, and after three minutes notifies to FACA the 60% certainty in the working connection, as documented in Listing 5.3. Afterwards, the user triggers authorization for the same IoT service as previously. However, his request has been denied (cf. Figure 5.15b) since the confidence did not satisfy the given access rule because the minimum required confidence has been set to 70%.

Listing 5.1: C-A access policy

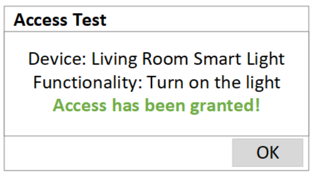
```
{
 {
  "_id" : ObjectId("607c4fcd54e0b501d0134f97"),
  "deviceName" : "Living Room Smart Light",
  "cloudDeviceTypeId" : "5f8600e103031223072051cd",
  "deviceTypeName" : "Phillips Hue Color",
  "function" : "Turn on the light",
  "rule" : {
        "accessRuleType" : "CONTEXT",
  "accessRule" : {
   "attributeName" : "Context.Connectivity.ConnectedToCloud",
   "operator" : "IS_TRUE",
   "accessRuleType" : "BOOLEAN",
        },
  "expectedCertaintyRule" : {
   "accessRuleType" : "NUMERIC",
   "expectedValue" : 70.0,
   "attributeName" : "Context.Connectivity.ConnectedToCloud",
   "operator" : "GREATER_THAN"
        },
  "attributeName" : "Context.Connectivity.ConnectedToCloud",
  "isAttributeValueTimeCritical" : false,
 },
 "priority" : 0,
 "enabled" : true
}
```

Listing 5.2: C-A value notification 1

```
{
   "attributeName":"Context
     .Connectivity.ConnectedToCloud",
   "contextType":"CONNECTIVITY",
   "attributeValue":"true",
   "certainty":100.0,
   "timestamp":[2021,4,20,16,00,
               49.521683900]
}
```

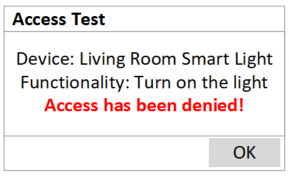
Listing 5.3: C-A value notification 2

```
{
   "attributeName":"Context
     .Connectivity.ConnectedToCloud",
   "contextType":"CONNECTIVITY",
   "attributeValue":"true",
   "certainty":60.0,
   "timestamp":[2021,4,20,16,03,
               29.843646900]
}
```


Access Test

Device: Living Room Smart Light
Functionality: Turn on the light
Access has been granted!

(a) Granted access due to the valid C-A agent's certainty


Access Test

Device: Living Room Smart Light
Functionality: Turn on the light
Access has been denied!

(b) Denied access due to the insufficient C-A agent's certainty

Figure 5.15: FACA access policies validation

5.5.2 *C-A Authorization Performance Analysis*

The selected evaluation scenarios target the impact of C-A access policies onto authorization latencies. This is achieved through a variable complexity of access policies: the amount of access rules with C-A attributes in a single access policy. Amounts of C-A per access policy used in scenarios are the same as for the FACA evaluations presented in Section 5.4.2: 1, 5, 10, 15, and 20 access rules. Other evaluation variables are fixed to simulate the Smart Home environment and also follow the ones presented in Table 5.8 for the *Policy complexity* evaluation aspect: five users, where each user sends 50 authorization requests during the test. Finally, the criticality of access policies plays a significant role in the overall performance, defining if the C-A attribute value will be retrieved from the database or C-A agent. Therefore, examining this performance aspect requires performance evaluation execution for both critical and non-critical access policies.

Non-critical access policies imply fetching C-A attribute values from FACA's database, which C-A agents previously notified. This allows FACA to retrieve all attribute values locally, without the need to contact C-A agents during authorization. Amounts of C-A attributes fetched from the database during authorization have been evaluated, resulting in the measurements presented in Figure 5.16. The presented results show that number of access rules with the C-A attribute does not increase with the access policy complexity. While E2E latency slightly grows, the scenario with 20 access policies shows that latency increase is not consistent, which can be influenced by the database's optimization procedures, such as data or query results caching.

The latency distribution for non-critical access policies presented in Table 5.12 provides further details concerning E2E latency. The majority of authorization requests (more than 92%) are executed within 105ms in each scenario. However, a slight increase in latency values for scenarios with 15 and 20 access rules is noticeable. Still, the noticed latency increase is negligible, meaning that it does not jeopardize the scalability of the presented C-A AC solution.

The second performance evaluation focuses on the execution of critical access policies. This requires each C-A attribute value to be fetched from the C-A agent before being validated using the XACML engine. For that purpose, three C-A agents are deployed, and attribute evaluation routes are evenly distributed between C-A attributes resulting in an even load on each C-A agent.

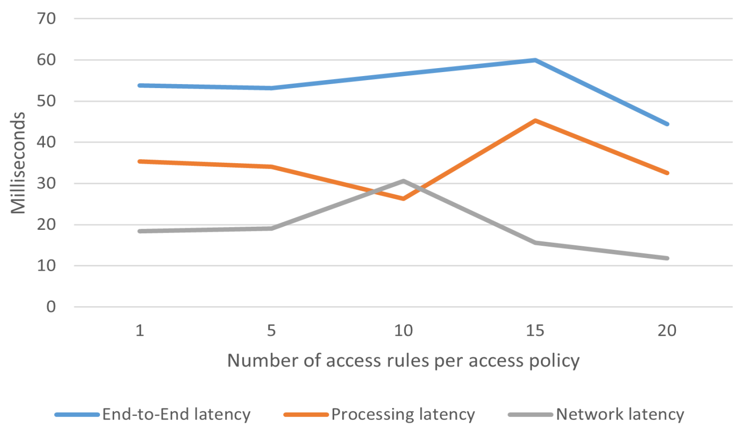


Figure 5.16: Average authorization latency - non-critical C-A attribute values

Table 5.12: E2E latency distribution in %- non-critical C-A attribute values

Latency limit [ms]	1 Access Rule	5 Access Rules	10 Access Rules	15 Access Rules	20 Access Rules
350.00	0.0	0.0	0.4	0.0	0.0
315.00	0.4	0.8	0.0	0.0	0.0
280.00	1.6	1.2	0.0	0.0	0.0
245.00	0.0	0.0	0.0	0.0	0.0
210.00	0.0	0.0	0.0	0.0	0.0
175.00	0.4	0.0	0.0	1.6	0.0
140.00	2.0	2.81	1.2	6.8	0.0
105.00	13.2	11.65	4.4	23.6	9.2
70.00	51.6	57.03	47.2	51.2	58.8
35.00	30.8	26.51	46.8	16.8	32.0

The average authorization latencies with critical C-A attributes are presented in Figure 5.17. While network latency remains even for all scenarios, both processing and E2E latency increase with the access rules number. The highest increase is between one and five access rules, that is from 121.8 ms to 259.29 ms, which results in 111% latency increase. This is mostly caused by the number of C-A agents that are contacted during the authorization. Namely, one access rule policy contacts one C-A agent, while five access rules are evenly distributed between C-A agents, contacting all three of them and causing network traffic for context information exchange. Afterwards, a steady increase in E2E and processing latencies from 5 to 15 access rules is noticeable, from 258.29 ms to 285.18 ms, resulting in an overall increase of 10%. The subsequent dramatic latency growth is between 15 and 20 access rules - around 31%. This indicates that the C-A agent request grouping proposed in Listing 4.27 enables a communication reduction between FACA and C-A agents during the authorization and partially hinders the latency growth with more complex access policies, but does not stop it completely.

The latencies distribution for critical C-A authorization provided in Table 5.13 speaks in favor of the results mentioned above. The required time to execute 95% of authorization requests indicates significant growth of the E2E latency. Whereas contacting one C-A agent in one access rule scenario requires less than 280 ms for most requests, in scenarios with 5 and 10 access rules the latency introduced through communication with three C-A agents increases to 420 ms. Furthermore, the same percentage of requests for authorization of 15 and 20 access rules requires 700ms, stating that the evaluated solution needs further optimization to be scalable in IoT systems with more complex access policies.

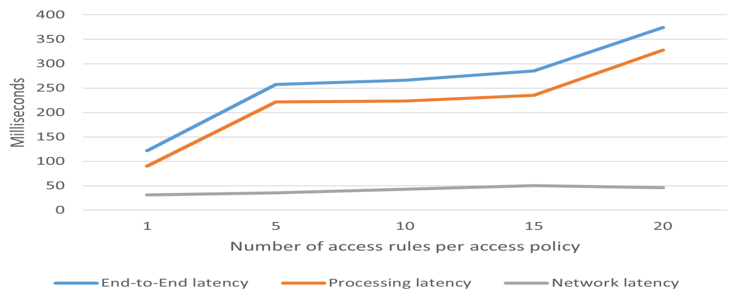


Figure 5.17: Average authorization latency - critical C-A attribute values

Table 5.13: E2E latency distribution in % - critical C-A attribute values

Latency limit [ms]	1 Access Rule	5 Access Rules	10 Access Rules	15 Access Rules	20 Access Rules
1400.00	0.0	0.0	0.0	0.4	0.0
1260.00	0.0	0.0	0.0	0.0	0.0
1120.00	0.0	0.0	0.0	0.0	0.0
980.00	0.0	0.0	0.0	0.0	0.8
840.00	0.0	0.0	0.0	1.2	1.6
700.00	0.0	0.0	0.0	2.8	8.4
560.00	0.0	2.4	4.8	6.0	20.4
420.00	4.0	12.4	14.8	25.2	42.4
280.00	21.2	74	74.8	62.4	26.4
140.00	74.8	11.2	5.6	2.0	0.0

5.5.3 *Context-Aware Authorization Performance Overhead*

The authorization process, which involves user and C-A attributes, requires fetching attribute values from multiple sources, as presented in Listing 4.27. Separated by attribute value type, these sources are:

- Session token for user attributes fetched from the digitally signed in-memory storage - JWS token;
- FACA's database for C-A attributes in non-critical access rule scenarios;
- AMQP-based attribute value retrieval from C-A agent for critical access rules.

Since attribute value sources differ in access times, the implications of all attribute value sources on processing and E2E latency have been evaluated. This is executed by comparing authorization latency measurements with C-A attributes documented in Section 5.5.2 with those using user attributes exclusively presented in Section 5.4.2, resulting in the C-A authorization's overhead.

The processing latency comparison for three attribute value sources is presented in Figure 5.18. In this diagram, the user and non-critical C-A attributes retrieval keeps comparable performances with no significant latency increases. The difference between the authorization with non-critical C-A attributes and with only user attributes varies between 5.26 ms or 25% increase for the scenario with 10 access rules and 23.98 ms or 113% for the scenario with 15 access rules. However, despite introducing the latency increase, authorization with non-critical C-A attributes shows no constant increase, indicating its scalability with a higher number of access rules in an access policy. In contrast, authorization latencies involving critical C-A attributes require an order of magnitude more time than the user attributes. The smallest overhead is present in the scenario with one access rule, in which the processing duration difference is 70.34 ms, which results in a 350% latency increase. Once the access policy contains more access rules, involving additional communication with C-A agents to fetch the latest C-A attribute values, processing latency increases to a maximal 307.55 ms or 1506%, representing this solution's significant overhead.

The E2E latencies comparison presented in Figure 5.19 follows the same patterns as the previous one. Non-critical and user attributes keep stable values, with a minimal increase of 6.44 ms or 17% for the scenario with 20 access rules and a maximal increase of 22.97ms or 68.2% for the scenario with 10 access rules. A decrease in percentage difference compared to the processing overhead is imposed by the network latency, which is almost the same for both scenarios and was not considered in the processing overhead evaluation. The authorization with critical C-A attributes introduces the significant E2E overhead since it increases the latency between 89.9 ms (281%) in the case of one access rule and 336.16 ms (884%) in the case of 20 access rules scenario. Significant overhead is mostly caused by the above-mentioned processing latency, which involves additional AMQP communication to fetch context information from C-A agents during the authorization.

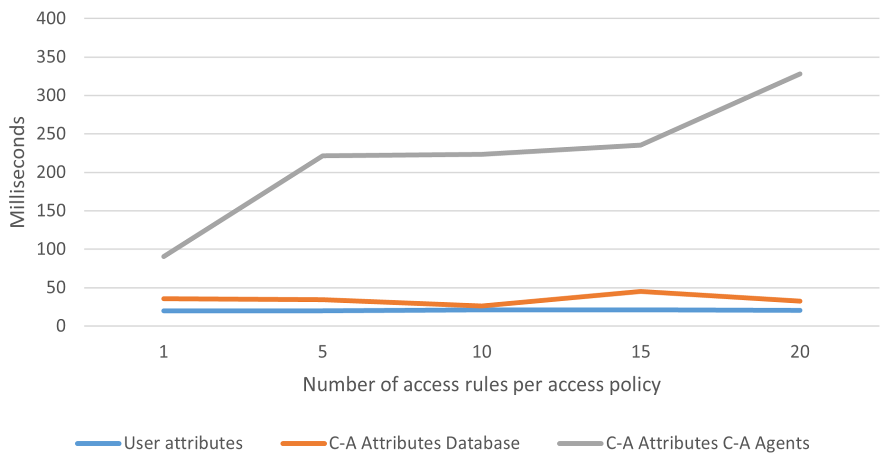


Figure 5.18: C-A authorization processing latency overhead

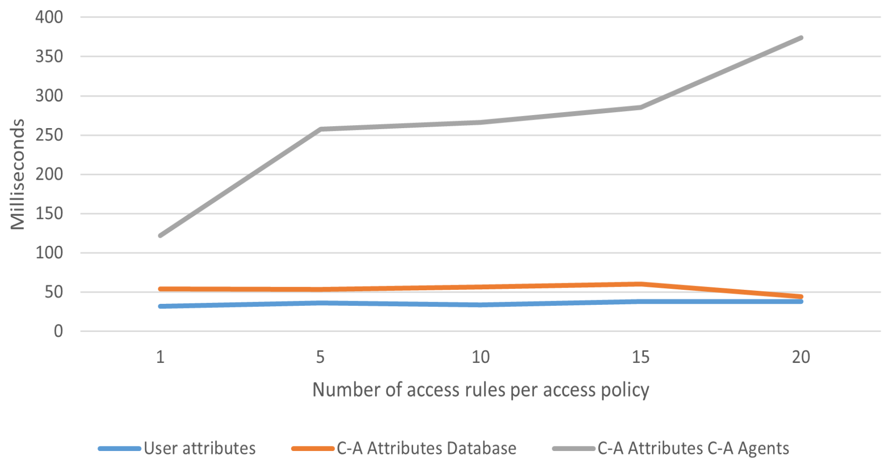


Figure 5.19: C-A authorization E2E latency overhead

5.6 Error Handling Procedures

The developed COSYLab framework provides FC-based IoT services by deploying them on the existing FNs in IoT networks. Availability and robustness of FNs impact the COSYLab components' overall stability. Due to those reasons, the fault-tolerance of the COSYLab represents a developed solution's critical aspect. Various software and hardware failure scenarios and their resolution within the COSYLab framework are discussed in this section.

Software failures can affect both CC and FC services. In most cases, these failures occur if CPU or RAM resources are depleted and make the affected service unresponsive until the service is restarted. Since the independence on CC services availability is one of the central premises for FC, all FC components within COSYLab are designed and developed to function despite complete CC services absence. For that reason, software failures of CC components do not introduce any downtime of FC services.

Observed FC components' software failures are divided into two categories:

1. The FC service's file system becomes corrupted;
2. The FC service becomes unresponsive.

A corrupted service's file system implies storing malformatted, not-usable data in the service's Docker container. This does not involve the data stored in MongoDB or RabbitMQ containers, but reflects on FC services' security material, such as public-key pairs, digital certificates, or offline-shared secrets. Resolving the corrupted file system requires restarting the Docker container with a clean file system and reinitializing the service's security material through the trust bootstrapping procedures presented in Section 3.2.2. Trust bootstrapping procedure in FTA differs from the ones in other FC components. Since FTA also exists in CTFTP, it must remain identifiable by TNTA despite corrupted security material, so that TNTA can revoke its digital certificate. This is resolved through the offline-shared secrets described in Figure 3.6, allowing FTA to rejoin CTFTP and TNTA to revoke FTA's previous certificate.

The solution for the failures from the second category is based on the Docker technology. Using Docker containers, services can be easily restarted, which recovers their availability. This approach implies service downtime until the Docker container has been restarted. The downtime can be minimized through a redundant service deployment - two or more deployed service instances in FC network, which can overtake the operations of the failed service instance.

Hardware failures occur once FN stops working due to a system error without the possibility to recover. This stops all deployed FC services deployed on the FN, with possible permanent data loss. Hardware failures' effects are minimized through the FC services redundancy mentioned above, improving the COSYLab services' stability and fault-tolerance. The COSYLab service redundancy does not resolve data loss issues. To prevent permanent data loss in MongoDB and RabbitMQ, the COSYLab relies on the clustering capabilities of these solutions. MongoDB offers data replication between multiple nodes[55] through the clusters deployed on one or multiple FNs. Similarly, exchanged AMQP messages between FC components are replicated within the RabbitMQ cluster[56], ensuring that messages will not be lost if a FN hardware failure occurs.

55 `https://www.mongodb.com/basics/clusters`, last access May 2, 2022
56 `https://www.rabbitmq.com/clustering.html`, last access May 2, 2022

6

Conclusions & Research Outlook

This dissertation tackled the challenge of designing and deploying security services close to the IoT networks' edge using Fog Computing (FC) capabilities. The designed and implemented solution - COSYLab - has been evaluated in the Smart Home environment using various Fog Node (FN) devices currently available on the market. The presented results focus on functional and performance aspects of the COSYLab, implying its application possibilities in IoT systems, leading to the creation of more secure and trustworthy IoT environments. In this chapter, the main results of the research conducted during this thesis are reflected, focusing on the main contributions, defined research questions, and future research and development of the COSYLab framework.

The presented thesis contributions revolve around the research questions outlined in Section 1.3, with the significant goal of advancing the state-of-the-art in creating trustworthy security services at the edge of IoT networks through the proposed messaging and data models. The main thesis contributions are categorized into three major research areas: FC, Trustworthy Networking (TN), and Context-Aware Access Control (C-A AC).

FC represents the underlying layer for the contributions in other research areas since it provides computing and storage resources for deploying TN, AC, and C-A services. The main problems solved by the contributions in the FC area are: (i) satisfying requirements for the development of FC-based IoT systems, (ii) deploying developed services, and (iii) providing performant FC-based IoT services. Addressing these problems led to the contributions with regard to the FC-based IoT systems design, implementation, and evaluation, presented in Sections 3.1, 4.3, and 5.1, respectively.

Section 3.1 provides a requirements analysis for building FC-based IoT solutions, focusing on the approaches for creating highly virtualized execution environments required for FC services deployment. Based on this analysis, the system design decisions are documented later in that section, providing guidelines for deploying trustworthy, standalone, and scalable FC-based IoT services. The structured, modularized FC services deployment model is presented in Section 4.3, providing insights for the COSYLab deployment along with required dependencies like networking, data storage, and credentials distribution. Lastly, in Section 5.1 the FC services feasibility and applicability evaluation is documented. The deployed services performances are presented concerning the resource (CPU, RAM) consumption and startup times on diverse FNs, resulting in the definition of computational resource requirements for deploying one or multiple FC services on the FN devices as significant output.

FC-based TN services provisioning implied the examination of various trust management models involving challenges in research areas like Key Management Protocols (KMP), Identity Management (IdM), and End-to-End (E2E) security. In addition, the solution design and implementation were heavily impacted by the nature of IoT devices like diversity or resource constraints and FC requirements for having operable FC services despite the unavailability of Cloud Servers. Resolving these challenges involved analysis of the state-of-the-art trust models (cf. Section 2.3.2), which served as a basis for the solution design, implementation, and evaluation, leading to the significant contributions concerning trust management in the FC-based IoT systems. The developed solution provides TN services for all actors in FC-based IoT systems - Things, users, and services. Achieving that resulted in a first contribution in the IdM area, i.e. designing the standard IdM schema for all actors presented in Section 3.2.1, maintaining the hierarchical organization of IoT system in the core identifier's fields. The second contribution area is represented through the trust management model enabling TN services provisioning independently of Cloud availability. This is achieved through splitting the Cloud - Fog - Thing continuum into Cloud-to-Fog Trust Plane (CTFTP) and Fog-to-Thing Trust Plane (FTTTP) and involving different KMPs within distributed TNDs (cf. Section 3.2.2). The developed KMPs, i.e. trust bootstraping, certificate validation, and certificate revocation can be adjusted to the current availability of the CC services and offer optimal trust management capabilities accordingly.

Since the applicability of the developed trust management models is constrained through the Things' computational capabilities, the third contribution area focuses on deploying E2E security in IoT. First, state-of-the-art trust management protocols and underlying encryption schemes like RSA, ECC, ECQV are evaluated through performance simulations against various factors occurring in the IoT environment, such as the Things' CPU and memory resources, network bandwidth, or number interconnected Things (cf. Section 4.5). The simulation results indicated that the best-performing trust management model is based on EC and X.509 certificates. Nevertheless, not all Things can perform asymmetric encryption, which requires rethinking PKI in IoT through the usage of FC. This resulted in the novel model presented in Section 3.2.3, offering (i) "best-effort" E2E security using security profiles that adjust to the Things' computational capabilities and (ii) possibilities for monitoring the Things' security capabilities during the IoT system lifecycle.

Further contributions in the TN are concerned with the solution implementation and evaluation. The proposed trust management model involves using (a) X.509 certificates, whose content corresponds to the proposed protocols, and (b) Tickets, representing a novel concept for a mutual authentication between Things, Users, FC, and CC services, as presented in Section 4.4. Lastly, the developed solution has been evaluated on multiple FN devices concerning its latency and scalability, leading to the results provided in Section 5.3 that prove the solution's feasibility and applicability in modern IoT systems.

The third significant contribution area is coupled with enabling C-A AC services using FC capabilities. Providing targeted services involved collecting and integrating context information in AC mechanisms, posing the following challenges: (i) a trustworthy AC services distribution and the maintenance of security policies consistency and (ii) embedding context information in security policies, focusing on the extension of the data collection and authorization procedures. Based on these challenges, multiple contributions have been provided.

Distributed AC is a well-researched IT area represented through protocols like OAuth2, OIDC, or SAML. However, these protocols are CC-focused and do not satisfy the FC requirement for FNs operability in offline scenarios. Due to that, this dissertation proposed a novel approach for the trustworthy distribution of AC mechanisms close to the IoT network edge. The proposed AC distribution model is based on two strategies: (1) trustworthy FC services distribution (cf. Section 3.2.2) and (2) synchronization procedures for the configuration of security policies presented in Section 3.3.2. Furthermore, the AC distribution model enables:

1. Users to manage and enforce the local security policies, leading to the improved control over their private data;
2. IoT platform administrators to maintain a consistent state of the security policies in the IoT platform through the synchronization of the access policies configuration.

Besides AC distribution, achieving C-A AC led to the contributions concerning the building of C-A solutions and their integration in AC. First, provisioning C-A AC required analyzing the state-of-the-art AC models presented in Section 2.4.2, resulting in choosing ABAC as the best approach for integrating context information in security policies. Second, the comprehensive analysis documented in Section 2.5 examines the established approaches for building C-A solutions concerning (a) context identification, (2) C-AS architectures, and (3) context information management and lifecycle. Findings from that analysis led to the design and implementation of the innovative model that integrates context information into security policies. The integration consists of the following aspects:

- A distributed software architecture allowing extensibility of the collected context information during the system's runtime. The architecture is based on the context information exchanges between FACA and C-A agents and extension of authorization procedures, as presented in Section 3.4.1.
- The introduction of the ontology-based data model for integrating context information into ABAC model (cf. Section 3.4.2);
- Creating a proof of concepts for the designed solution using three different context-aware sources: location, users' behavior, and internet connectivity, as described in Section 3.4.3.

Finally, the performance and functionality of the implemented C-A AC solution has been evaluated, considering multiple factors: security policy complexity, IoT users number, and authorization requests concurrency. The performance evaluation has been conducted separately for AC, without C-A factors. AC without C-A factors results presented in Section 5.4 prove the applicability of the developed solution in Smart Home environments and provide insights on the bottlenecks that should be addressed to extend the solution's application in other IoT application domains. The performance evaluation with C-A factors documented in Section 5.5 indicates that the proposed C-A integration model can be applied in IoT environments with a certain latency overheads, varying from 50% to 620%, depending on the context information integration strategy.

During this dissertation, different research questions influenced the design of the presented solution for trustworthy, FC-based C-A AC in an IoT environment - COSYLab. The research questions listed in Section 1.3 are now revisited and answered, outlining the research impact of this dissertation.

(RQ 1) How can decentralized management of access control be achieved in a Fog computing environment?

> **(RQ 1.1)** Which techniques and technologies can be used in order to keep a consistent state of an access control framework in the hierarchical IoT organization: Cloud Server – Fog Node - Things?
>
> **(RQ 1.2)** What kind of trust connections between authentication and authorization entities need to be achieved to minimize attack surfaces on end devices with regard to access control?

Computational capabilities provided through FN can be beneficial for deploying AC mechanisms at the IoT networks' edge. This occurs through the deployment of ACC as local agents in the IoT environment, which maintains trust relationships with Cloud Server during the system's lifecycle. Despite numerous present protocols that suit this purpose (e.g., OAuth2 or SAML), the FC requirement for having fully operable FC services independent of Cloud availability hinders the application these protocols. To satisfy the abovementioned FC requirement, COSYLab proposes ACC deployment on FNs with all functionalities for providing AAA mechanisms - IdM, security policies management, authorization, etc.

To respond to RQ 1 and RQ 1.1, the ACC component in COSYLab named FACA maintains trust relationships during its lifetime using PKI. PKI enables certificate management and mutual authentication between FC and CC services (cf. Section 3.2.2). Through that, each FACA instance and its operations (e.g., user authentication or request authorization) is considered trustworthy across IoT platforms. Moreover, FACA's security policy management occurs exclusively on locally deployed FNs, binding access rules to the present Things' functionalities. To keep the consistent state of the security policies on each FACA, configuration synchronization between ACAM and FACA occurs whenever ACAM is available, ensuring that only safelisted Things' functionalities can be authorized by FACA (cf. Section 3.3.2).

Ensuring trust connections between authentication and authorization entities mentioned in RQ 1.2 requires them to ensure their trustworthiness through the mutual authentication. COSYLab employs digital certificates to authenticate FC services and users, resulting in digitally signed user session tokens - JWS, containing user information as attributes (cf. Section 4.6.1). For that reason, user attributes provided in JWS issued by FACA are considered trustworthy and can be securely validated by security policies during authorization.

(RQ 2) How can Fog computing be used for improving Identity management and authentication in IoT systems?

> **(RQ 2.1)** How can computational power on the edge of the network be used for improving scalability of automatic (e.g., PKI or Web of Trust) authentication procedures?
>
> **(RQ 2.2)** How can Fog nodes be efficiently and effectively used as secure storage of identities for resource-constrained devices and providing mutual authentication in IoT environments?

Procedures for IdM and authentication mentioned in RQ2 introduce extensive computational load on the devices due to the encryption procedures. As FC utilizes deployed hardware resources in IoT networks to ensure sufficient computing capabilities, it can bridge the gap for using acknowledged security protocols between resource-constrained Things and remote Cloud services. This involves deploying various trust management and IdM models, offloading tasks from central Cloud Servers to local FNs. The deployment of security protocols through FC services must be followed by appropriate security mechanisms, ensuring their trustworthiness by satisfying standard KMP requirements as analyzed in Section 2.3.

The deployed FC services can further create mutual trust relationships between IoT services, users, and Things, creating isolated TNDs. This increases the identities and credentials manageability through the divide-and-conquer strategy to offload the computationally demanding tasks from Cloud Servers to FNs. Moreover, it leads to the increased scalability, robustness, and extensibility of the authentication procedures referred to in RQ 2.1. Through that, FNs represent a central point for managing security operations like KMP, IdM, or certificate management in local TND - KMP, IdM, cert management. Achieving this requires rethinking and extending state-of-the-art KMP procedures between different TNDs, as examined in Section 3.2.2.

Furthermore, to answer RQ 2.2, possibilities for using FNs as identity and credentials storage for resource-constrained Things have been examined. As traditional protocols for encryption-based mutual authentication heavily rely on digital signatures, their applicability cannot be guaranteed on all IoT devices.

The main reason for that are computational requirements for the asymmetric encryption. Having FN with sufficient resources for supporting an acknowledged trust management model close to the Things allows their utilization as gatekeepers, that is, trust mediators between Things and the rest of the IoT environment. Through that, computationally demanding operations required for mutual authentication can be offloaded to FNs. In COSYLab, FTP components offer these services, hosting IdM and certificate management processes for Things while establishing "best-effort" trust relationships based on security profiles and E2E security with the Things (cf. Section 3.2.3). Security profiles are determined mainly by Things computational capabilities and negotiated with FTP during trust bootstrapping procedures. When applying this approach, networking and FTP processing have to be taken into consideration. Results presented in Section 5.3.2 prove that even less powerful FN devices can host FTP service for the Smart Home environment but introduce a significant latency. For that reason, it is essential to consider the time constraints for the developed IoT application domain once deploying this kind of TN service on FN devices.

(RQ 3) How can context-awareness be incorporated in access control of a Fog computing based IoT system?

> **(RQ 3.1)** Which factors can influence access rights validation and adjustments in a context-aware IoT environments?
>
> **(RQ 3.2)** How can these factors be embedded in today's access control models?

As a response to question RQ 3, Section 2.5 provides a comprehensive analysis of the building blocks for C-A solutions, followed by the thorough analysis of design steps required to build the C-A AC system presented in Section 3.4. Essentially, integration of context information in AC has two major building blocks: data collection and authorization procedures extension. Data collection needs to allow mechanisms for ACC to consume collected context information and store it for future use during authorization. It is essential to distribute context information collection across multiple components and support system characteristics like flexibility, extensibility, trustworthiness, and scalability to scale with the types and amount of context information easily.

Data collection is tightly bound to RQ 3.1, questioning which factors can be considered relevant context information for AC in IoT. As the purpose of numerous IoT application domains strongly differs, finding common context factors set for all of them is hardly achievable. Despite this fact, context information is categorized through multiple criteria (cf. Section 2.5.2): (i) observed entities (Things, users, and places), (ii) context type (logical or physical). These criteria, along with the downsizing of C-AS requirements to a particular IoT application domain, represent good practice for identifying context factors. Following this strategy, COSYLab incorporated location, connectivity, and user behavior as representative context information for Smart Home use-case (cf. Section 3.4.3).

Finally, collected context information has to be integrated with AC procedures, as mentioned in RQ 3.2. Achieving this requires using context information in the sense that users can configure security policies involving context information. C-A security policy management requires an appropriate AC schema, for which a thorough analysis is provided in Section 3.3.1. Based on the chosen AC schema, context information provisioning that satisfies C-A solutions' characteristics requires distributed architecture, based on the common message exchanges (cf. Section 3.4.1) and data-model (cf. Section 3.4.2). Configured C-A security policies are applied during authorization, involving the collected context information. As the collected context information can be considered obsolete during security policy validation, it is recommended to allow fetching current context values directly during authorization, guaranteeing their up-to-dateness. As evaluated in Section 5.5.2, this involves a significant increase of the authorization delay. For that reason, during building C-A AC solutions a trade-off between context information up-to-dateness and authorization duration has to be met.

This dissertation proposed multiple models, schemas, and protocols for establishing TN and C-A AC services using FC. Despite the achieved contributions, several challenges remained that are out of scope of this thesis. The following paragraphs summarize the possible future research directions for the solution developed in this dissertation.

AC solution through ACAM and FACA modules offers AC distribution from Cloud to Fog. Future plans for these modules involve the further distribution of AAA mechanisms to the IoT network's edge. This would allow additional robustness and minimize the centrality of the current solution. The main effort for this improvement requires rethinking and extending the current synchronization procedures and applied ABAC model.

The second planned research direction is aimed towards trust establishment between FTTTP TNDs through FC-based trust federations. This would further decrease the independence of FTA and FTP on CC services and allow FC-based IoT networks to exchange their service despite Cloud Servers' unavailability. For this purpose, the proposed PKI schema should be extended with additional trust bootstrapping and KMP protocols.

Regarding the C-AS area, development and integration of further context information use-cases are considered beneficial for potential improvement of the current data- and integration model. Within the Smart Home environment, various information sources and their applicability in AC will be examined, offering more possibilities for AC automation.

Finally, the application of COSYLab in other IoT application domains like Smart Building or Smart City is planned to analyze further the feasibility of the developed protocols and models, as well as the scalability and robustness of the developed solution.

Appendices

A

Docker Deployment

This appendix provides details on the deployment of COSYLab components. Deployment is based on the Docker Compose, which enables building configuration bundles used for installing and running multi-container Docker environments. Inside its configuration bundles, Docker Compose allows specification of the: deployed services, their configurations, dependencies, and network setup.

As presented in Listing A.1, CC services rely on MongoDB for data storage, as well as Nginx[57] server to proxy incoming traffic to the CC services - ACAM, TNTA, and CSWA (lines 3-22). Moreover, additional servers are deployed to enable integration between the developed services (lines 23-35). Those servers enable service discovery and configuration deployment, enabling communication between TNTA and ACAM. Finally, the developed CC services are deployed, as described in lines 36-71.

Docker Compose configuration (cf. Listing A.2) for FC services is split into core FC services - Fog Controller, FTA and FACA (lines 23-68) along with configuration for MongoDB and AMQP-broker deployment (lines 3-22). Additional FC services can be deployed through separated Docker Compose configuration files, as presented in Listing A.3 for FTP and Listing A.2 for CCAA.

57 https://www.nginx.com/, last access May 2, 2022

```
version: '3.5'
services:
    reverse-proxy:
        image: nginx
        container_name: reverse-proxy
        hostname: reverse-proxy
        volumes:
            - ./nginx/config:/etc/nginx
            - ./nginx/ssl:/etc/ssl/private
        ports:
            - 80:80
        links:
            - cosylab-cswa
            - cosylab-acam
            - cosylab-tnta
    cloud-mongodb:
        image: mongo
        container_name: cloud-mongodb
        restart: always
        environment:
            MONGO_INITDB_ROOT_USERNAME: ${MONGO_USERNAME}
            MONGO_INITDB_ROOT_PASSWORD: ${MONGO_PASSWORD}
    cosylab-cloud-discovery-server:
        image: ignjatov90/cosylab-cloud-discovery-server
        container_name: cosylab-cloud-discovery-server
        hostname: cosylab-cloud-discovery-server
    cosylab-cloud-config-server:
        image: ignjatov90/cosylab-cloud-config-server
        container_name: cosylab-cloud-config-server
        hostname: cosylab-cloud-config-server
        links:
            - cosylab-cloud-discovery-server
        depends_on:
            cosylab-cloud-discovery-server:
                condition: service_healthy
    cosylab-cswa:
        image: ignjatov90/cosylab-cloud-cswa
        container_name: cosylab-cswa
        hostname: cosylab-cswa
    cosylab-acam:
        image: ignjatov90/cosylab-cloud-acam
        container_name: cosylab-acam
        hostname: cosylab-acam
        environment:
            COSYLAB_MONGO_USER: ${MONGO_USERNAME}
            COSYLAB_MONGO_PASS: ${MONGO_PASSWORD}
            COSYLAB_ACAM_USER: ${ACAM_USERNAME}
            COSYLAB_ACAM_PASS: ${ACAM_PASSWORD}
        links:
            - cloud-mongodb
            - cosylab-cloud-discovery-server
            - cosylab-cloud-config-server
            - cosylab-tnta
        depends_on:
            cosylab-cloud-config-server:
                condition: service_healthy
    cosylab-tnta:
        image: ignjatov90/cosylab-cloud-tnta
        container_name: cosylab-tnta
        hostname: cosylab-tnta
        environment:
            COSYLAB_MONGO_USER: ${MONGO_USERNAME}
            COSYLAB_MONGO_PASS: ${MONGO_PASSWORD}
            COSYLAB_TNTA_KEYSTORE_PASS: ${TNTA_KEYSTORE_PASS}
        links:
            - cloud-mongodb
            - cosylab-cloud-discovery-server
            - cosylab-cloud-config-server
        depends_on:
            cosylab-cloud-config-server:
                condition: service_healthy
```

Listing A.1: Docker Compose CC service bundle

```yaml
version: '3.5'
services:
    fog_rabbitmq:
        image: rabbitmq:3-management
        container_name: fog_rabbitmq
        environment:
            RABBITMQ_DEFAULT_USER: ${RABBITMQ_USERNAME}
            RABBITMQ_DEFAULT_PASS: ${RABBITMQ_PASSWORD}
        ports:
            - 5672:5672
        healthcheck:
            test: rabbitmq-diagnostics -q ping
            interval: 5s
            timeout: 2s
            retries: 10
    fog_mongodb:
        image: mongo
        container_name: fog_mongodb
        restart: always
        environment:
            MONGO_INITDB_ROOT_USERNAME: ${MONGO_USERNAME}
            MONGO_INITDB_ROOT_PASSWORD: ${MONGO_PASSWORD}
    fog_ctrl:
        image: ignjatov90/fog_ctrl
        container_name: fog_ctrl
        ports:
            - 80:8080
        links:
            - fog_mongodb
            - fog_rabbitmq
        environment:
            COSYLAB_FOG_CTRL_FACA_URL: http://fog_faca:5000
            COSYLAB_FOG_CTRL_HA_URL: http://host.docker.internal:8123/api
        depends_on:
            fog_rabbitmq:
                condition: service_healthy
    fog_fta:
        image: ignjatov90/fog_fta
        container_name: fog_fta
        links:
            - fog_mongodb
            - fog_rabbitmq
        volumes:
          - ftavolume:/keystore
        environment:
            COSYLAB_FTA_AS: ${FTA_AS}
            COSYLAB_FTA_IS: ${FTA_IS}
            COSYLAB_FTA_KEYSTORE_PASS: ${FTA_KEYSTORE_PASSWORD}
        depends_on:
            fog_rabbitmq:
                condition: service_healthy
    fog_faca:
        image: ignjatov90/fog_faca
        container_name: fog_faca
        links:
            - fog_mongodb
            - fog_rabbitmq
        volumes:
          - facavolume:/keystore
        environment:
            COSYLAB_FACA_KEYSTORE_PASS: ${FTA_KEYSTORE_PASSWORD}
            COSYLAB_FACA_ACAM_URL: http://ni-bakk.cosy.univie.ac.at/
        depends_on:
            fog_rabbitmq:
                condition: service_healthy
volumes:
  ftavolume:
  facavolume:
```

Listing A.2: Docker Compose core FC service bundle

```
version: '3.5'
services:
    fog_ftp:
        image: ignjatov90/fog_ftp
        container_name: fog_ftp
        volumes:
          - ftpvolume:/keystore
        environment:
            COSYLAB_MONGO_HOST: ${MONGO_HOST}
            COSYLAB_MONGO_USER: ${MONGO_USERNAME}
            COSYLAB_MONGO_PASS: ${MONGO_PASSWORD}
            COSYLAB_RABBIT_HOST: ${RABBITMQ_HOST}
            COSYLAB_RABBIT_USER: ${RABBITMQ_USERNAME}
            COSYLAB_RABBIT_PASS: ${RABBITMQ_PASSWORD}
            COSYLAB_FTP_KEYSTORE_PASS:  ${FTP_KEYSTORE_PASSWORD}
volumes:
  ftpvolume:
```

Listing A.3: Docker Compose FTP service

```
version: '3.5'
services:
    fog_ccaa:
        image: ignjatov90/fog_ccaa
        container_name: fog_ccaa
        volumes:
          - ccaavolume:/keystore
        environment:
            COSYLAB_MONGO_HOST: ${MONGO_HOST}
            COSYLAB_MONGO_USER: ${MONGO_USERNAME}
            COSYLAB_MONGO_PASS: ${MONGO_PASSWORD}
            COSYLAB_RABBIT_HOST: ${RABBITMQ_HOST}
            COSYLAB_RABBIT_USER: ${RABBITMQ_USERNAME}
            COSYLAB_RABBIT_PASS: ${RABBITMQ_PASSWORD}
            COSYLAB_FTP_KEYSTORE_PASS:  ${CCAA_KEYSTORE_PASSWORD}
volumes:
        ccaavolume:
```

Listing A.4: Docker Compose C-A agent example

B

PKI Simulation Configuration

This appendix contains the configuration files used for PKI scalability simulations. The provided configuration files are used for simulating following PKI schemas and KMPs:

1. RSA with Diffie-Hellman key exchange and CRL in Listings B.1 and B.2;
2. RSA with Diffie-Hellman key exchange and OCSP in Listings B.3 and B.4;
3. EC with Diffie-Hellman key exchange and CRL in Listings B.6 and B.7;
4. EC with Diffie-Hellman key exchange and OCSP in Listings B.8 and B.9;
5. EC with Qu-Vanstone key exchange and CRL in Listings B.11 and B.12;
6. EC with Qu-Vanstone key exchange and OCSP in Listings B.13 and B.14.

Furthermore, various certificate management approaches are configured as follows: (1) Listing B.5 provides global configuration for RSA X.509v3 certificates, (2) Listing B.10 contains configuration for ECC X.509v3 certificates, and (3) Listing B.15 configures ECC implicit certificates. Finally, simulated devices are configured using Listing B.16.

```
"create_L": {
 "1": {
  "name": "Start","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": { "nextStep": 2, "sendTo": "t" }}
 },
 "2": {
  "name": "Gen Key", "ram_strain": 2648, "time": 83996,
  "next_step": {"Option_1": {"nextStep": 3, "sendTo": "t"}}
 },
 "3": {
  "name": "Gen CSR", "ram_strain": 260, "time": 3081,
  "payload": {"size": 1375},
  "next_step": {"Option_1": {"nextStep": 4, "sendTo": "p"}}
 },
 "4": {
  "name": "Create Certificate", "ram_strain": 130, "time": 290,
  "payload": {"size": 4830},
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Save Cert", "ram_strain": 144, "time": 10}
},
"create_I": {
 "1": {
  "name": "Start","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "t"}}
 },
 "2": {
  "name": "Gen Key","ram_strain": 2648,"time": 40000,
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "t"}}
 },
 "3": {
  "name": "Gen CSR","ram_strain": 260,"time": 124,
  "payload": {"size": 1375},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "p"}}
 },
 "4": {
  "name": "Create Certificate", "ram_strain": 130,"time": 120,
  "payload": {"size": 3220},
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Save Cert", "ram_strain": 144, "time": 10}
},
"expire_revoke": {
 "1": {
  "name": "Expire on Me", "ram_strain": 144, "time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "p"}}
 },
 "2": {
  "name": "Expire on I","ram_strain": 144,"time": 3,
  "payload": {"size": 100, "mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "p"}}
 },
 "3": {
  "name": "Expire on R","ram_strain": 144, "time": 3,
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "s"}}
 },
 "4": {
  "name": "Return L", "ram_strain": 144, "time": 3,
  "next_step": { "Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Finish","ram_strain": 1,"time": 1}
}
```

Listing B.1: RSA CRL create and expire lifecycles

```
"validate": {
 "1": {
  "name": "Request Cert","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "a"}}
 },
 "2": {
  "name": "Sending Cert","ram_strain": 144,"time": 10,
  "payload": {"size": 4830,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "s"}}
 },
 "3": {
  "name": "Check Root Cert","ram_strain": 144,"time": 42,
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "t"}}
 },
 "4": {
  "name": "Is Valid Root Cert","ram_strain": 1,"time": 1,"check_valid_aff": true,
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "t"}}
 },
 "5": {
  "name": "Check CRL","ram_strain": 144,"time": 10,"condition": "has_Crl",
  "next_step": {"Option_1": {"nextStep": 6,"sendTo": "t"},
               "Option_2": {"nextStep": 9,"sendTo": "t"}}
 },
 "6": {
  "name": "Request CRL", "ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 7,"sendTo": "p"}}
 },
 "7": {
  "name": "Send CRL","ram_strain": 96,"time": 10,
  "next_step": {"Option_1": {"nextStep": 8,"sendTo": "s"}},
  "payload": {"size": 100,"mult_with": "num_revo_cert"}},
 "8": {
  "name": "Save CRL","ram_strain": 96,"time": 1,
  "next_step": {"Option_1": {"nextStep": 9,"sendTo": "t"}}
 },
 "9": {
  "name": "Loop CRL","ram_strain": 100,"time": 1,"check_revoke_list": "CRL"
        "next_step": {"Option_1": {"nextStep": 10,"sendTo": "t"}},
 },
 "10": {
  "name": "Is not Revoked","ram_strain": 1,"time": 1,"check_valid_aff": true
  "next_step": {"Option_1": {"nextStep": 11,"sendTo": "t"}},
 },
 "11": {
  "name": "Set Alice Context and send","ram_strain": 374,"time": 440,
  "payload": {"size": 360,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 12,"sendTo": "a"}}
 },
 "12": {
  "name": "Setting Bob Context","ram_strain": 372,"time": 440,
  "next_step": {"Option_1": {"nextStep": 13,"sendTo": "t"}}
 },
 "13": {
  "name": "Bob Read Alice Values and gen Secret","ram_strain": 374, "time": 250,
  "payload": {"size": 116,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 14,"sendTo": "s"}}
 },
 "14": {
  "name": "Alice reading Bob and gen Key","ram_strain": 360,"time": 250
 }
}
```

Listing B.2: RSA CRL validation lifecycle

```
"create_L": {
 "1": {
  "name": "Start","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "t"}}
 },
 "2": {
  "name": "Gen Key","ram_strain": 2648,"time": 83996,
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "t"}}
 },
 "3": {
  "name": "Gen CSR","ram_strain": 260,"time": 3081,
  "payload": {"size": 1375},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "p"}}
 },
 "4": {
  "name": "Create Certificate","ram_strain": 1090,"time": 290,
  "payload": {"size": 4830},
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {
  "name": "Save Cert","ram_strain": 144,"time": 10
 }
},
"create_I": {
 "1": {
  "name": "Start","ram_strain": 1,"time": 1,
        "next_step": {"Option_1": {"nextStep": 2,"sendTo": "t"}}
 },
 "2": {
  "name": "Gen Key","ram_strain": 2648,"time": 40000,
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "t"}}
 },
 "3": {
  "name": "Gen CSR","ram_strain": 260,"time": 124,
  "payload": {"size": 1375},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "p"}}
 },
 "4": {
  "name": "Create Certificate","ram_strain": 130,"time": 120,
  "payload": {"size": 3220},
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {
  "name": "Save Cert","ram_strain": 144,"time": 10
 }
},
"expire_revoke": {
 "1": {
  "name": "Expire on Me","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "p"}}
 },
 "2": {
  "name": "Expire on I","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "p"}}
 },
 "3": {
  "name": "Expire on R","ram_strain": 144,"time": 3,
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "s"}}
 },
 "4": {
  "name": "Return L","ram_strain": 144,"time": 3,
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {
  "name": "Finish","ram_strain": 1,"time": 1
 }
}
```

Listing B.3: RSA OCSP create and expire lifecycles

```
"validate": {
 "1": {
  "name": "Request Cert","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "a"}}
 },
 "2": {
  "name": "Sending Cert","ram_strain": 144,"time": 10,
  "payload": {"size": 4830,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "s"}}
 },
 "3": {
  "name": "Check Root Cert","ram_strain": 144,"time": 42,
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "t"}}
 },
 "4": {
  "name": "Is Valid Root Cert","ram_strain": 1,"time": 1,"check_valid_aff": true,
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "t"}}
 },
 "5": {
  "name": "Check CRL","ram_strain": 144,"time": 10,
  "next_step": {"Option_1": {"nextStep": 6,"sendTo": "t"}}
 },
 "6": {
  "name": "OSCP Request","ram_strain": 1,"time": 10,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 7,"sendTo": "r"}}
 },
 "7": {
  "name": "Check OSCP","ram_strain": 96,"time": 10,"check_revoke_list": "OSCP",
  "next_step": {"Option_1": {"nextStep": 8,"sendTo": "s"}},
  "payload": {"size": 100,"mult_with": "num_revo_cert"}},
 "8": {
  "name": "Check Valid","ram_strain": 10,"time": 10,"check_valid_aff": true,
  "next_step": {"Option_1": {"nextStep": 9,"sendTo": "t"}}
 },
 "9": {
  "name": "Set Alice Context and send","ram_strain": 374,"time": 440,
  "payload": {"size": 360,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 10,"sendTo": "a"}}
 },
 "10": {
  "name": "Setting Bob Context","ram_strain": 440,"time": 240,
  "next_step": {"Option_1": {"nextStep": 11,"sendTo": "t"}}
 },
 "11": {
  "name": "Bob Read Alice Values and gen Secret","ram_strain": 374,"time": 250,
  "payload": {"size": 116,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 12,"sendTo": "s"}}
 },
 "12": {
  "name": "Alice reading Bob and gen Key","ram_strain": 360,"time": 250
 }
}
```

Listing B.4: RSA OCSP validation lifecycle

```
"const_data": {
 "crl_entry_size": 100, "asym_key_size": 384, "sym_key_size": 128,
 "session_key_recheck": 60, "cert_size": 1610
}
```

Listing B.5: RSA global configuration

```
"create_L": {
 "1": {
  "name": "Start","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "t"}}
 },
 "2": {
  "name": "Gen Key","ram_strain": 2648,"time": 240,
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "t"}}
 },
 "3": {
  "name": "Gen CSR","ram_strain": 260,"time": 248,
  "payload": {"size": 500},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "p"}}
 },
 "4": {
  "name": "Create Certificate","ram_strain": 130,"time": 120,
  "payload": {"size": 1500},
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Save Cert","ram_strain": 144,"time": 10}
},
"create_I": {
 "1": {
  "name": "Start","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "t"}}
 },
 "2": {
  "name": "Gen Key","ram_strain": 2648,"time": 120,
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "t"}}
 },
 "3": {
  "name": "Gen CSR","ram_strain": 260,"time": 124,
  "payload": {"size": 500},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "p"}}
 },
 "4": {
  "name": "Create Certificate","ram_strain": 130,"time": 60,
  "payload": {"size": 1000},
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Save Cert","ram_strain": 144,"time": 10}
 },
"expire_revoke": {
 "1": {
  "name": "Expire on Me","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "p"}}
 },
 "2": {
  "name": "Expire on I","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "p"}}
 },
 "3": {
  "name": "Expire on R","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "s"}}
 },
 "4": {
  "name": "Return L","ram_strain": 144,"time": 3,
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Finish","ram_strain": 1,"time": 1}
}
```

Listing B.6: ECC CRL create and expire lifecycles

```
"validate": {
 "1": {
  "name": "Request Cert","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "a"}}
 },
 "2": {
  "name": "Sending Cert","ram_strain": 144,"time": 10,
  "payload": {"size": 1500,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "s"}}
 },
 "3": {
  "name": "Check Root Cert","ram_strain": 144,"time": 42,
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "t"}}
 },
 "4": {
  "name": "Is Valid Root Cert","ram_strain": 1,"time": 1,"check_valid_aff": true,
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "t"}}
 },
 "5": {
  "name": "Check CRL","ram_strain": 144,"time": 10,
  "next_step": {"Option_1": {"nextStep": 6,"sendTo": "t"}}
 },
 "6": {
  "name": "Request CRL","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 7,"sendTo": "p"}}
 },
 "7": {
  "name": "Send CRL","ram_strain": 96,"time": 10,
  "payload": {"size": 100,"mult_with": "num_revo_cert"},
  "next_step": {"Option_1": {"nextStep": 8,"sendTo": "s"}}
 },
 "8": {
  "name": "Save CRL","ram_strain": 96,"time": 1,
  "next_step": {"Option_1": {"nextStep": 9,"sendTo": "t"}}
 },
 "9": {
  "name": "Loop CRL","ram_strain": 100,"time": 1,"check_revoke_list": "CRL",
  "next_step": {"Option_1": {"nextStep": 10,"sendTo": "t"}}
 },
 "10": {
  "name": "Is not Revoked","ram_strain": 1,"time": 1,"check_valid_aff": true,
  "next_step": {"Option_1": {"nextStep": 11,"sendTo": "t"}}
 },
 "11": {
  "name": "Set Alice Context and send","ram_strain": 209,"time": 372,
  "payload": {"size": 360,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 12,"sendTo": "a"}}
 },
 "12": {
  "name": "Setting Bob Context","ram_strain": 372,"time": 209,
  "next_step": {"Option_1": {"nextStep": 13,"sendTo": "t"}}
 },
 "13": {
  "name": "Bob Read Alice Values and gen Secret","ram_strain": 116,"time": 216,
  "payload": {"size": 116,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 14,"sendTo": "s"}}
 },
 "14": {
  "name": "Alice reading Bob and gen Key","ram_strain": 116,"time": 218
 }
}
```

Listing B.7: ECC CRL validation lifecycle

```
"create_L": {
 "1": {
  "name": "Start","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "t"}}
 },
 "2": {
  "name": "Gen Key","ram_strain": 2648,"time": 240,
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "t"}}
 },
 "3": {
  "name": "Gen CSR","ram_strain": 260,"time": 248,
  "payload": {"size": 500},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "p"}}
 },
 "4": {
  "name": "Create Certificate","ram_strain": 130,"time": 120,
  "payload": {"size": 1500},
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Save Cert","ram_strain": 144,"time": 10}
},
"create_I": {
 "1": {
  "name": "Start","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "t"}}
 },
 "2": {
  "name": "Gen Key","ram_strain": 2648,"time": 120,
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "t"}}
 },
 "3": {
  "name": "Gen CSR","ram_strain": 260,"time": 124,
  "payload": {"size": 500},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "p"}}
 },
 "4": {
  "name": "Create Certificate","ram_strain": 130,"time": 120,
  "payload": {"size": 1000},
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Save Cert","ram_strain": 144,"time": 10}
},
"expire_revoke": {
 "1": {
  "name": "Expire on Me","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "p"}}
 },
 "2": {
  "name": "Expire on I","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "p"}}
 },
 "3": {
  "name": "Expire on R","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "s"}}
 },
 "4": {
  "name": "Return L","ram_strain": 144,"time": 3,
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Finish","ram_strain": 1,"time": 1}
}
```

Listing B.8: ECC OCSP create and expire lifecycles

```
"validate": {
 "1": {
  "name": "Request Cert","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "a"}}
 },
 "2": {
  "name": "Sending Cert","ram_strain": 144,"time": 10,
  "payload": {"size": 1500,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "s"}}
 },
 "3": {
  "name": "Check Root Cert","ram_strain": 144,"time": 10,
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "t"}}
 },
 "4": {
  "name": "Is Valid Root Cert","ram_strain": 1,"time": 1,"check_valid_aff": true,
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "t"}}
 },
 "5": {
 "name": "Query OSCP Request","ram_strain": 1,"time": 10,
 "payload": {"size": 100,"mult_with": "1"},
 "next_step": {"Option_1": {"nextStep": 6,"sendTo": "r"}}
 },
 "6": {
  "name": "Query OSCP","ram_strain": 69,"time": 1,"check_revoke_list": "OSCP",
  "next_step": {"Option_1": {"nextStep": 7,"sendTo": "s"}}
 },
 "7": {
  "name": "Check Valid","ram_strain": 1,"time": 1,"check_valid_aff": true,
  "next_step": {"Option_1": {"nextStep": 8,"sendTo": "t"}}
 },
 "8": {
  "name": "Set Alice Context and send","ram_strain": 209,"time": 372,
  "payload": {"size": 360,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 9,"sendTo": "a"}}
 },
 "9": {
  "name": "Setting Bob Context","ram_strain": 372,"time": 209,
  "next_step": {"Option_1": {"nextStep": 10,"sendTo": "t"}}
 },
 "10": {
  "name": "Bob Read Alice Values and gen Secret","ram_strain": 116,"time": 216,
  "payload": {"size": 116,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 11,"sendTo": "s"}}
 },
 "11": {
  "name": "Alice reading Bob and gen Key","ram_strain": 116,"time": 218
 }
}
```

Listing B.9: ECC OCSP validation lifecycle

```
"const_data": {
 "crl_entry_size": 100,"asym_key_size": 32,"sym_key_size": 128,
 "session_key_recheck": 60,"cert_size": 500
}
```

Listing B.10: ECC global configuration

```
"create_L": {
 "1": {
  "name": "Load Param from ECQV","ram_strain": 24,"time": 10,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "t"}}
 },
 "2": {
  "name": "Gen Key","ram_strain": 2300,"time": 240,
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "t"}}
 },
 "3": {
  "name": "Gen CSR","ram_strain": 260,"time": 248,
  "payload": {"size": 290},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "p"}}
 },
 "4": {
  "name": "Create Certificate","ram_strain": 130,"time": 120,
  "payload": {"size": 984},
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Save Cert and Finish Priv Key","ram_strain": 244,"time": 20}
},
"create_I": {
 "1": {
  "name": "Load Param from ECQV","ram_strain": 24,"time": 10,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "t"}}
 },
 "2": {
  "name": "Gen Key","ram_strain": 2300,"time": 120,
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "t"}}
 },
 "3": {
  "name": "Gen CSR","ram_strain": 260,"time": 124,
  "payload": {"size": 290},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "p"}}
 },
 "4": {
  "name": "Create Certificate","ram_strain": 130,"time": 60,
  "payload": {"size": 656},
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Save Cert and Finish Priv Key","ram_strain": 244,"time": 10}
},
"expire_revoke": {
 "1": {
  "name": "Expire on Me","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "p"}}
 },
 "2": {
  "name": "Expire on I","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "p"}}
 },
 "3": {
  "name": "Expire on R","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "s"}}
 },
 "4": {
  "name": "Return L","ram_strain": 144,"time": 3,
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Finish","ram_strain": 1,"time": 1}
}
```

Listing B.11: ECQV CRL create and expire lifecycles

```
"validate": {
 "1": {
  "name": "Request Cert","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "a"}}
 },
 "2": {
  "name": "Sending Cert","ram_strain": 144,"time": 10,
  "payload": {"size": 984},
  "next_step": {"Option_1": "nextStep": 3,"sendTo": "s"}}
 },
 "3": {
  "name": "Extract Public Key","ram_strain": 200,"time": 50,
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "t"}}
 },
 "4": {
  "name": "Check Root Cert","ram_strain": 144,"time": 42,
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "t"}}
 },
 "5": {
  "name": "Is Valid Root Cert","ram_strain": 1,"time": 1,"check_valid_aff": true,
  "next_step": {"Option_1": {"nextStep": 6,"sendTo": "t"}}
 },
 "6": {
  "name": "Check CRL","ram_strain": 144,"time": 10,
  "next_step": {"Option_1": {"nextStep": 7,"sendTo": "t"}}
 },
 "7": {
  "name": "Request CRL","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 8,"sendTo": "r"}}
 },
 "8": {
  "name": "Send CRL","ram_strain": 96,"time": 10,
  "payload": {"size": 100,"mult_with": "num_revo_cert"},
  "next_step": {"Option_1": {"nextStep": 9,"sendTo": "s"}}
 },
 "9": {
  "name": "Save CRL","ram_strain": 96,"time": 1,
  "next_step": {"Option_1": {"nextStep": 10,"sendTo": "t"}}
 },
 "10": {
  "name": "Loop CRL","ram_strain": 100,"time": 1,"check_revoke_list": "CRL",
  "next_step": {"Option_1": {"nextStep": 11,"sendTo": "t"}}
 },
 "11": {
  "name": "Is not Revoked","ram_strain": 1,"time": 1,"check_valid_aff": true,
  "next_step": {"Option_1": {"nextStep": 12,"sendTo": "t"}}
 },
 "12": {
  "name": "Set Alice Context and send","ram_strain": 209,"time": 372,
  "payload": {"size": 360,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 13,"sendTo": "a"}}
 },
 "13": {
  "name": "Setting Bob Context","ram_strain": 372,"time": 209,
  "next_step": {"Option_1": {"nextStep": 14,"sendTo": "t"}}
 },
 "14": {
  "name": "Bob Read Alice Values and gen Secret","ram_strain": 116,"time": 216,
  "payload": {"size": 116,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 15,"sendTo": "s"}}
 },
 "15": {"name": "Alice reading Bob and gen Key","ram_strain": 116,"time": 218}
}
```

Listing B.12: ECQV CRL validation lifecycle

```
"create_L": {
 "1": {
  "name": "Load Param for ECQV","ram_strain": 24,"time": 10,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "t"}}
 },
 "2": {
  "name": "Gen Key","ram_strain": 2648,"time": 240,
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "t"}}
 },
 "3": {
  "name": "Gen CSR","ram_strain": 260,"time": 248,
  "payload": {"size": 290},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "p"}}
 },
 "4": {
  "name": "Create Certificate","ram_strain": 130,"time": 120,
  "payload": {"size": 984},
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Save Cert and finish priv Key","ram_strain": 244,"time": 20}
},
"create_I": {
 "1": {
  "name": "Load Param from ECQV","ram_strain": 24,"time": 10,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "t"}}
 },
 "2": {
  "name": "Gen Key","ram_strain": 2300,"time": 120,
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "t"}}
 },
 "3": {
  "name": "Gen CSR","ram_strain": 260,"time": 124,
  "payload": {"size": 290},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "p"}}
 },
 "4": {
  "name": "Create Certificate","ram_strain": 130,"time": 60,
  "payload": {"size": 656},
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Save Cert and Finish Priv Key","ram_strain": 244,"time": 10}
},
"expire_revoke": {
 "1": {
  "name": "Expire on Me","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "p"}}
 },
 "2": {
  "name": "Expire on I","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "p"}}
 },
 "3": {
  "name": "Expire on R","ram_strain": 144,"time": 3,
  "payload": {"size": 100,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "s"}}
 },
 "4": {
  "name": "Return L","ram_strain": 144,"time": 3,
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "s"}}
 },
 "5": {"name": "Finish","ram_strain": 1,"time": 1}
}
```

Listing B.13: ECQV OCSP create and expire lifecycles

```
"validate": {
 "1": {
  "name": "Request Cert","ram_strain": 1,"time": 1,
  "next_step": {"Option_1": {"nextStep": 2,"sendTo": "a"}}
 },
 "2": {
  "name": "Sending Cert","ram_strain": 144,"time": 10,
  "payload": {"size": 1500,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 3,"sendTo": "s"}}
 },
 "3": {
  "name": "Extract Public Key","ram_strain": 200,"time": 50,
  "next_step": {"Option_1": {"nextStep": 4,"sendTo": "t"}}
 },
 "4": {
  "name": "Check Root Cert","ram_strain": 144,"time": 10,
  "next_step": {"Option_1": {"nextStep": 5,"sendTo": "t"}}
 },
 "5": {
  "name": "Is Valid Root Cert","ram_strain": 1,"time": 1,"check_valid_aff": true
  "next_step": {"Option_1": {"nextStep": 6,"sendTo": "t"}}
 },
 "6": {
  "name": "Query OSCP Request","ram_strain": 1,"time": 10,
  "next_step": {"Option_1": {"nextStep": 7,"sendTo": "r"}},
  "payload": {"size": 100,"mult_with": "1"}
 },
 "7": {
  "name": "Query OSCP","ram_strain": 69,"time": 1,"check_revoke_list": "OSCP"
  "next_step": {"Option_1": {"nextStep": 8,"sendTo": "s"}},
 },
 "8": {
  "name": "Check Valid","ram_strain": 1,"time": 1,"check_valid_aff": true
        "next_step": {"Option_1": {"nextStep": 9,"sendTo": "t"}},
 },
 "9": {
  "name": "Set Alice Context and send","ram_strain": 209,"time": 372,
  "payload": {"size": 360,"mult_with": "1"},
  "next_step": {"Option_1": {"nextStep": 10,"sendTo": "a"}}
 },
 "10": {
  "name": "Setting Bob Context","ram_strain": 372,"time": 209,
  "next_step": {"Option_1": {"nextStep": 11,"sendTo": "t"}}
 },
 "11": {
  "name": "Bob Read Alice Values and gen Secret","ram_strain": 116,"time": 216,
  "next_step": {"Option_1": {"nextStep": 12,"sendTo": "s"}},
  "payload": {"size": 116,"mult_with": "1"}
 },
 "12": {"name": "Alice reading Bob and gen Key","ram_strain": 116,"time": 218}
}
```

Listing B.14: ECQV OCSP validation lifecycle

```
"const_data": {
 "name": "ECQV","crl_entry_size": 100,"asym_key_size": 32,
 "sym_key_size": 128,"session_key_recheck": 60,"cert_size": 328
},
```

Listing B.15: ECQV global configuration

```
{
  "GLOBAL": {
    "create_lower": 30000,
    "create_upper": 90000,
    "validate_lower": 30000,
    "validate_upper": 90000,
    "revoke_lower": 30000,
    "revoke_upper": 120000,
    "validation_timeout": 40000,
    "max_ms" : 1000
  },
  "R_NODE": {
    "max_thread": 8,
    "max_storage": 9000000,
    "max_ram": 10000,
    "max_tries": 1,
    "wtask_seed_lower": 1000,
    "wtask_seed_upper": 2000
  },
  "I_NODE": {
    "max_thread": 4,
    "max_storage": 2621440,
    "max_ram": 655360,
    "max_tries": 20,
    "wtask_seed_lower": 30000,
    "wtask_seed_upper": 60000
  },
  "L_NODE": {
    "max_thread": 1,
    "max_storage": 1310720,
    "max_ram": 327680,
    "max_tries": 20,
    "wtask_seed_lower": 60000,
    "wtask_seed_upper": 120000
  }
}
```

Listing B.16: Global configuration - time constraints and PKI nodes

List of Figures

List of Tables

List of Listings

Acronyms

AAA	Authentication, Authorization, and Accounting
ABAC	Attribute-Based Access Control
AC	Access Control
ACAM	Access Control Agents Management
ACC	Access Control Center
AEKMP	Asymmetric Encryption-based KMP
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
BCAA	Behavior Context-Awareness Agent
C-A	Context-Awareness
C-A AC	Context-Aware Access Control
C-AS	Context-Aware System
CA	Certificate Authority
CAPBAC	Capability-Based Access Control
CC	Cloud Computing
CCAA	Connectivity Context-Awareness Agent
CIA	Confidentiality, Integrity, and Availability

CN	Constrained Node
CRL	Certificate Revocation List
CSR	Certificate Signing Request
CTFTP	Cloud-to-Fog Trust Plane
DAC	Discretionary Access Control
DH	Diffie-Hellman
DNS	Domain Name System
DTLS	Datagram Transport Layer Security
E2E	End-to-End
EC	Edge Computing
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
EDC	Edge Data Center
FACA	Fog Access Control Agent
FC	Fog Computing
FIdM	Federated IdM
FN	Fog Node
FTA	Fog Trust Anchor
FTP	Fog Trust Provider
FTTTP	Fog-to-Thing Trust Plane
HSM	Hardware Security Module
IBAC	Identity-Based Access Control
IdM	Identity Management

IdMS	Identity Management System
IdP	Identity Provider
IoT	Internet of Things
IP	Internet Protocol
IPSec	Internet Protocol Security
IT	Information Technology
JWS	JSON Web Signature
JWT	JSON Web Token
KDC	Key Distribution Center
KMP	Key Management Protocol
LATBAC	Lattice-Based Access Control
LCAA	Location Context-Awareness Agent
MAC	Mandatory Access Control
MCC	Mobile Edge Computing
MDACS	Multi-Domain Access Control Systems
MEC	Mobile Cloud Computing
OCSP	Online Certificate Status Protocol
OFRA	OpenFog Reference Architecture
OWL	Web Ontology Language
PA	Policy Anchor
PAP	Policy Administration Point
PDP	Policy Decision Point
PEM	Privacy Enhanced Mail

PEP	Policy Enforcement Point
PIP	Policy Information Point
PKI	Public-Key Infrastructure
QoS	Quality of Service
QV	Qu-Vanstone
RBAC	Role-Based Access Control
RDF	Resource Description Framework
REST	Representational State Transfer
RFID	Radio-Frequency Identification
RSA	Rivest–Shamir–Adleman
SAML	Security Assertion Markup Language
SEKMP	Symmetric Encryption-based KMP
SPoF	Single-Point-of-Failure
SSL	Secure Sockets Layer
SSO	Single Sign-On
TA	Trust Anchor
TCI	Trust Center Infrastructure
TN	Trustworthy Networking
TND	Trustworthy Networking Domain
TNTA	Trustworthy Network Trust Anchor
TPM	Trusted Platform Module
URL	Uniform Resource Locator
WWW	World Wide Web

Bibliography

[22a] *IEEE 1934-2018 - IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing*. June 2018 (accessed May 2, 2022). URL: https://standards.ieee.org/standard/1934-2018.html.

[22b] *Industrial Internet of Things Volume G4: Security Framework*. Sept. 2016 (accessed May 2, 2022). URL: https://www.iiconsortium.org/pdf/IIC_PUB_G4_V1.00_PB.pdf.

[22c] *ISO/IEC 24760-1:2019 IT Security and Privacy — A framework for identity management — Part 1: Terminology and concepts*. May 2019 (accessed May 2, 2022). URL: https://www.iso.org/standard/77582.html.

[22d] *ISO/IEC 7498-1:1994 Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model*. Nov. 1994 (accessed May 2, 2022). URL: https://www.iso.org/standard/20269.html.

[22e] *Mobile Edge Computing(MEC); Framework and Reference Architecture*. Mar. 2018 (accessed May 2, 2022). URL: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf.

[22f] *Multi-access Edge Computing(MEC); Framework and Reference architecture*. Mar. 2018 (accessed May 2, 2022). URL: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf.

[22g] *OpenFog Reference Architecture for Fog Computing.* Feb. 2017 (accessed May 2, 2022). URL: https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf.

[22h] *Optimizing Latency and Bandwidth for AWS Traffic.* Apr. 2016 (accessed May 2, 2022). URL: https://aws.amazon.com/blogs/startups/optimizing-latency-and-bandwidth-for-aws-traffic/.

[22i] *Security Assertion Markup Language (SAML) V2.0 Technical Overview.* Mar. 2008 (accessed May 2, 2022). URL: http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html.

[22j] *Transitive Trust Enrollment for Constrained Devices, Internet Draft draft-jennings-core-transitive-trust-enrollment-01.* Oct. 2012 (accessed May 2, 2022). URL: https://tools.ietf.org/html/draft-jennings-core-transitive-trust-enrollment-01.

[22k] *X.1250 : Baseline capabilities for enhanced global identity management and interoperability.* Sept. 2009 (accessed May 2, 2022). URL: https://www.itu.int/rec/T-REC-X.1250-200909-I.

[22l] *X.800 : Security architecture for Open Systems Interconnection for CCITT applications.* Mar. 1991 (accessed May 2, 2022). URL: https://www.itu.int/rec/T-REC-X.800-199103-I.

[22m] *Y.2720 : NGN identity management framework.* Jan. 2009 (accessed May 2, 2022). URL: https://www.itu.int/rec/T-REC-Y.2720-200901-I.

[22n] *Y.3052 : Overview of trust provisioning for information and communication technology infrastructures and services.* Mar. 2017 (accessed May 2, 2022). URL: https://www.itu.int/rec/T-REC-Y.3052-201703-I.

[22o] *Y.3053 : Framework of trustworthy networking with trust-centric network domains.* Jan. 2018 (accessed May 2, 2022). URL: https://www.itu.int/rec/T-REC-Y.3053-201801-I.

[22p] *Y.4000 : Overview of the Internet of things.* June 2012 (accessed May 2, 2022). URL: https://www.itu.int/rec/T-REC-Y.4000/en.

[AAC16] U. Alegre, J. C. Augusto, and T. Clark. "Engineering context-aware systems and applications: A survey". In: *Journal of Systems and Software* 117 (Feb. 2016), pp. 55–83. ISSN: 0164-1212.

[Abo+99a] Abowd et al. "Towards a Better Understanding of Context and Context-Awareness". In: *HUC 1999: Handheld and Ubiquitous Computing*. Nov. 1999, pp. 304–307.

[Abo+99b] Abowd et al. "Towards a Better Understanding of Context and Context-Awareness". In: *Handheld and Ubiquitous Computing*. Jan. 1999.

[Ada+05] C. Adams, S. Farrell, T. Kause, and T. Mononen. *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*. RFC 4210 (Proposed Standard). RFC. Updated by RFC 6712. Fremont, CA, USA: RFC Editor, Sept. 2005. URL: https://www.rfc-editor.org/rfc/rfc4210.txt.

[AIM10] L. Atzori, A. Iera, and G. Morabito. "The Internet of Things: A survey". In: *Computer Networks* 54.15 (Oct. 2010), pp. 2787–2805. ISSN: 1389-1286.

[Ali09] M. Ali. "Green Cloud on the Horizon". In: *Cloud Computing*. Ed. by M. G. Jaatun, G. Zhao, and C. Rong. Springer Berlin Heidelberg, 2009, pp. 451–459. ISBN: 978-3-642-10665-1.

[And11] J. Andress. "Chapter 3 - Authorization and Access Control". In: *The Basics of Information Security*. Ed. by J. Andress. Syngress, June 2011. Chap. 3, pp. 33–49. ISBN: 978-1-59749-653-7.

[ARC18] M. Ammar, G. Russello, and B. Crispo. "Internet of Things: A survey on the security of IoT frameworks". In: *Journal of Information Security and Applications* 38 (Feb. 2018), pp. 8–27. ISSN: 2214-2126. URL: http://www.sciencedirect.com/science/article/pii/S2214212617302934.

[Bah+12] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan. "Advancing the State of Mobile Cloud Computing". In: *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*. MCS '12. ACM, 2012, pp. 21–28.

[Bar22] E. Barker. *NIST Special Publication 800-57 Part 1 Revision 4, Recommendation for Key Management, Part 1: General.* Tech. rep. 800-57. National Institute of Standards and Technology, Jan. 2016 (accessed May 2, 2022). URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf.

[BD03] L. Barkhuus and A. Dey. "Is context-aware computing taking control away from the user? Three levels of interactivity examined". In: *UbiComp 2003: Ubiquitous Computing*. Jan. 2003, pp. 149–156.

[BD22] E. Barker and Q. Dang. *Recommendation for Key Management Part 3: Application-Specific Key Management Guidance*. Tech. rep. National Institute of Standards and Technology, Jan. 2015 (accessed May 2, 2022). URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf.

[BDR07a] M. Baldauf, S. Dustdar, and F. Rosenberg. "A Survey on Context-Aware Systems". In: *Int. J. Ad Hoc Ubiquitous Comput.* 2.4 (June 2007), pp. 263–277. ISSN: 1743-8225.

[BDR07b] M. Baldauf, S. Dustdar, and F. Rosenberg. "A Survey on Context-aware systems". In: *International Journal of Ad Hoc and Ubiquitous Computing* 2.4 (June 2007), pp. 263–277. ISSN: 1743-8225.

[Bec+14] M. Beck, M. Werner, S. Feld, and T. Schimper. "Mobile Edge Computing: A Taxonomy". In: Jan. 2014.

[Bel+12] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini. "A Survey of Context Data Distribution for Mobile Ubiquitous Systems". In: *ACM Computing Surveys* 44.4 (Aug. 2012), pp. 1–45. ISSN: 0360-0300.

[Bel+19] P. Bellavista, J. Berrocal, A. Corradi, S. K. Das, L. Foschini, and A. Zanni. “A survey on fog computing for the Internet of Things”. In: *Pervasive and Mobile Computing* 52 (Mar. 2019), pp. 71–99. ISSN: 1574-1192.

[Bet+10] C. Bettini et al. “A survey of context modelling and reasoning techniques”. In: *Pervasive and Mobile Computing* 6.2 (June 2010). Context Modelling, Reasoning and Management, pp. 161–180. ISSN: 1574-1192.

[BFA05] A. Bunningen, L. Feng, and P. Apers. “Context for Ubiquitous Data Management”. In: *International Workshop on Ubiquitous Data Management.* Apr. 2005, pp. 17–24.

[Bia+10] G. Bianchi, A. Capossele, A. Mei, and C. Petrioli. “Flexible key exchange negotiation for wireless sensor networks”. In: *WiNTECH ’10: Proceedings of the fifth ACM international workshop on Wireless network testbeds, experimental evaluation and characterization* (Sept. 2010), pp. 55–62.

[BKC20] C. Bormann, A. Keranen, and C.Gomez. *Terminology for Constrained-Node Networks.* RFC 7228bis (Informational). RFC. Fremont, CA, USA: RFC Editor, Mar. 2020. URL: https://tools.ietf.org/html/draft-bormann-lwig-7228bis-06.

[BMW11] D. Beimborn, T. Miletzki, and S. Wenzel. “Platform as a Service (PaaS)”. In: *Wirtschaftsinformatik* 53.6 (Dec. 2011), pp. 371–375.

[BMZ13] M. Bafandehkar, R. Mahmod, and M. H. Zurina. “Comparison of ECC and RSA Algorithm in Resource Constrained Devices”. In: *2013 International Conference on IT Convergence and Security, ICITCS 2013.* Dec. 2013, pp. 1–3.

[Bon+12a] R. Bonetto, N. Bui, V. Lakkundi, A. Olivereau, A. Serbanati, and M. Rossi. “Secure communication for smart IoT objects: Protocol stacks, use cases and practical examples”. In: *2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM).* June 2012, pp. 1–7.

[Bon+12b] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. "Fog Computing and Its Role in the Internet of Things". In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. MCC '12. ACM, 2012, pp. 13–16.

[Bor+18] C. Bormann, S. Lemay, H. Tschofenig, K. Hartke, B. Silverajan, and B. Raymor (Ed.) *CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets*. RFC 8323 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Feb. 2018. URL: https://www.rfc-editor.org/rfc/rfc8323.txt.

[Bor14] E. Borgia. "The Internet of Things vision: Key features, applications and open issues". In: *Computer Communications* 54 (Dec. 2014), pp. 1–31. ISSN: 0140-3664.

[BPM16] Z. B. Babovic, J. Protic, and V. Milutinovic. "Web Performance Evaluation for Internet of Things Applications". In: *IEEE Access* 4 (Oct. 2016), pp. 6974–6992. ISSN: 2169-3536.

[Bro96] P. J. Brown. "The Stick-e Document: a Framework for Creating Context-aware Applications". In: *Proceedings of EP"96, Palo Alto*. Jan. 1996, pp. 259–272.

[BTC08] A. M. Bernardos, P. Tarrio, and J. R. Casar. "A data fusion framework for context-aware mobile services". In: *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*. Aug. 2008, pp. 606–613.

[Cal+07] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. *OpenPGP Message Format*. RFC 4880 (Proposed Standard). RFC. Updated by RFC 5581. Fremont, CA, USA: RFC Editor, Nov. 2007. URL: https://www.rfc-editor.org/rfc/rfc4880.txt.

[Cam22] M. Campagna. *ECMQV-ECQV Cipher Suites for Transport Layer Security (TLS)*. July 2010 (accessed May 2, 2022). URL: https://tools.ietf.org/pdf/draft-campagna-tls-ecmqv-ecqv-01.pdf.

[CBE00] D. G. Cholewka, R. A. Botha, and J. H. P. Eloff. “A Context-sensitive Access Control Model and Prototype Implementation”. In: *Information Security for Global Information Infrastructures.* Jan. 2000, pp. 341–350.

[ČH18] A. Čolaković and M. Hadžialić. “Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues”. In: *Computer Networks* 144 (July 2018), pp. 17–39. ISSN: 1389-1286.

[Cha09] D. W. Chadwick. “Federated Identity Management”. In: *Foundations of Security Analysis and Design V: FOSAD 2007/2008/2009 Tutorial Lectures.* Ed. by A. Aldini, G. Barthe, and R. Gorrieri. Springer Berlin Heidelberg, Aug. 2009, pp. 96–120. ISBN: 978-3-642-03829-7. URL: https://doi.org/10.1007/978-3-642-03829-7_3.

[Che03] H. Chen. “An Intelligent Broker Architecture for Context-Aware Systems”. In: *Adjunct Proceedings of UbiComp* (Feb. 2003).

[Cho+03] S. Chokhani, W. Ford, R. Sabett, C. Merrill, and S. Wu. *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework.* RFC 3647 (Informational). RFC. Fremont, CA, USA: RFC Editor, Nov. 2003. URL: https://www.rfc-editor.org/rfc/rfc3647.txt.

[CK00] G. Chen and D. Kotz. *A Survey of Context-Aware Mobile Computing Research.* Tech. rep. Dartmouth College, Computer and Information Systems Dept, Nov. 2000.

[CLC15] J. Chen, Y. Liu, and Y. Chai. “An Identity Management Framework for Internet of Things”. In: *2015 IEEE 12th International Conference on e-Business Engineering.* Oct. 2015, pp. 360–364.

[CMF12] E. Conrad, S. Misenar, and J. Feldman. “Chapter 2 - Domain 1: Access Control”. In: *CISSP Study Guide (Second Edition).* Ed. by E. Conrad, S. Misenar, and J. Feldman. Second Edition. Syngress, 2012, pp. 9–62. ISBN: 978-1-59749-961-3.

[CMJ15] B. Campbell, C. Mortimore, and M. Jones. *Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants.* RFC 7522 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, May 2015. URL: https://www.rfc-editor.org/rfc/rfc7522.txt.

[Coo+08a] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC 5280 (Proposed Standard). RFC. Updated by RFCs 6818, 8398, 8399. Fremont, CA, USA: RFC Editor, May 2008. URL: https://www.rfc-editor.org/rfc/rfc5280.txt.

[Coo+08b] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC 5280 (Proposed Standard). RFC. Updated by RFCs 6818, 8398, 8399. Fremont, CA, USA: RFC Editor, May 2008. URL: https://www.rfc-editor.org/rfc/rfc5280.txt.

[COO13] V. Coskun, B. Ozdenizci, and K. Ok. "A survey on Near Field Communication (NFC) technology". In: *Wireless Personal Communications* 71 (Aug. 2013), pp. 2259–2294. ISSN: 1572-834X.

[Cus10] M. Cusumano. "Cloud Computing and SaaS As New Computing Platforms". In: *Commun. ACM* 53.4 (Apr. 2010), pp. 27–29. ISSN: 0001-0782.

[Das+16] P. K. Das, S. Narayanan, N. K. Sharma, A. Joshi, K. Joshi, and T. Finin. "Context-Sensitive Policy Based Security in Internet of Things". In: *IEEE International Conference on Smart Computing (SMARTCOMP).* May 2016, pp. 1–6.

[DAS01] A. Dey, G. Abowd, and D. Salber. "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications".

In: *Human-Computer Interaction* 16.2 (Apr. 2001), pp. 97–166. ISSN: 0737-0024.

[Dey+01] A. Dey, G. Kortuem, D. Morse, and A. Schmidt. "Situated Interaction and Context-Aware Computing". In: *Personal and Ubiquitous Computing - PUC* 5.1 (Jan. 2001), pp. 1–3. ISSN: 1617-4909.

[Dey98] A. Dey. "Context-Aware Computing: The CyberDesk Project". In: (Jan. 1998).

[DH17] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification.* RFC 8200 (Internet Standard). RFC. Fremont, CA, USA: RFC Editor, July 2017. URL: https://www.rfc-editor.org/rfc/rfc8200.txt.

[Dou04] P. Dourish. "What We Talk about When We Talk about Context". In: *Personal Ubiquitous Comput.* 8.1 (Feb. 2004), pp. 19–30. ISSN: 1617-4909.

[EA13] D. Eastlake 3rd and J. Abley. *IANA Considerations and IETF Protocol and Documentation Usage for IEEE 802 Parameters.* RFC 7042 (Best Current Practice). RFC. Fremont, CA, USA: RFC Editor, Oct. 2013. URL: https://www.rfc-editor.org/rfc/rfc7042.txt.

[EJK19] S. van Engelenburg, M. Janssen, and B. Klievink. "Designing context-aware systems: A method for understanding and analysing context in practice". In: *Journal of Logical and Algebraic Methods in Programming* 103 (Feb. 2019), pp. 79–104. ISSN: 2352-2208.

[Fan+12] X. Fang, S. Misra, G. Xue, and D. Yang. "Smart Grid — The New and Improved Power Grid: A Survey". In: *IEEE Communications Surveys Tutorials* 14.4 (Apr. 2012), pp. 944–980. ISSN: 2373-745X.

[Fan+17] X. Fan, F. Susan, W. R. Long, and S. Li. "Security Analysis of Zigbee". In: *MIT.edu.* 2017.

[Far+18] B. Farahani, F. Firouzi, V. Chang, M. Badaroglu, N. Constant, and K. Mankodiya. “Towards fog-driven IoT eHealth: Promises and challenges of IoT in medicine and healthcare”. In: *Future Generation Computer Systems* 78 (Jan. 2018), pp. 659–676. ISSN: 0167-739X.

[FF14] R. Falk and S. Fries. “Managed Certificate Whitelisting – A Basis for Internet of Things Security in Industrial Automation Applications”. In: *The Eighth International Conference on Emerging Security Information, Systems and Technologies – SECURWARE 2014*. Nov. 2014.

[FLR13] N. Fernando, S. W. Loke, and W. Rahayu. “Mobile cloud computing: A survey”. In: *Future Generation Computer Systems* 29.1 (Jan. 2013), pp. 84–106. ISSN: 0167-739X.

[Gar+15] P. Garcia Lopez et al. “Edge-centric Computing: Vision and Challenges”. In: *SIGCOMM Comput. Commun. Rev.* 45.5 (Sept. 2015), pp. 37–42. ISSN: 0146-4833.

[GMS15] J. Granjal, E. Monteiro, and J. Sá Silva. “Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues”. In: *IEEE Communications Surveys and Tutorials* (July 2015), pp. 1–1. ISSN: 1553-877X.

[GPR13] S. Gusmeroli, S. Piccione, and D. Rotondi. “A capability-based security approach to manage access control in the Internet of Things”. In: *Mathematical and Computer Modelling* 58 (Sept. 2013), pp. 1189–1205. ISSN: 0895-7177.

[GQ18] B. Gupta and M. Quamara. “An identity based access control and mutual authentication framework for distributed cloud computing services in IoT environment using smart cards”. In: *Procedia Computer Science* 132 (May 2018), pp. 189–197. ISSN: 1877-0509.

[Gre01] S. Greenberg. “Context as a Dynamic Construct”. In: *Human–Computer Interaction* 16.2-4 (Dec. 2001), pp. 257–268.

[Gus02] R. Gustavsen. “Condor - an application framework for mobility-based context-aware applications”. In: (Jan. 2002).

[GZV11] G. Guo, J. Zhang, and J. Vassileva. “Improving PGP Web of Trust through the Expansion of Trusted Neighborhood”. In: *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*. Vol. 1. Aug. 2011, pp. 489–494.

[Had+19] H. Haddadpajouh, A. Dehghantanha, R. Parizi, M. Aledhari, and H. Karimipour. “A survey on internet of things security: Requirements, challenges, and solutions”. In: *Internet of Things* (Nov. 2019). ISSN: 2542-6605.

[Har12] D. Hardt (Ed.) *The OAuth 2.0 Authorization Framework*. RFC 6749 (Proposed Standard). RFC. Updated by RFC 8252. Fremont, CA, USA: RFC Editor, Oct. 2012. URL: https://www.rfc-editor.org/rfc/rfc6749.txt.

[HI04] K. Henricksen and J. Indulska. “A software engineering framework for context-aware pervasive computing”. In: *Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. Proceedings of the*. 2004, pp. 77–86.

[HIR08] P. Hu, J. Indulska, and R. Robinson. “An Autonomic Context Management System for Pervasive Computing”. In: *2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*. May 2008, pp. 213–223.

[HK89] S. Hares and D. Katz. *Administrative Domains and Routing Domains: A model for routing in the Internet*. RFC 1136 (Informational). RFC. Fremont, CA, USA: RFC Editor, Dec. 1989. URL: https://www.rfc-editor.org/rfc/rfc1136.txt.

[HNZ16] D. A. Ha, K. T. Nguyen, and J. K. Zao. “Efficient Authentication of Resource-Constrained IoT Devices Based on ECQV Implicit Certificates and Datagram Transport Layer Security Protocol”. In: *Proceedings of the Seventh Symposium on Information and Communication Technology*. 2016, pp. 173–179. URL: https://doi.org/10.1145/3011077.3011108.

[Hof+03] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger. “Context-awareness on mobile devices - the hydrogen approach”. In: *36th Annual Hawaii International Conference on System Sciences*. Jan. 2003.

[Hu+22] V. Hu et al. *Guide to Attribute Based Access Control (ABAC) Definition and Considerations.* Tech. rep. 800-162. National Institute of Standards and Technology, Jan. 2014 (accessed May 2, 2022). URL: `http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-162.pdf`.

[Hu16] F. Hu. *Security and Privacy in Internet of Things (IoTs): Models, Algorithms, and Implementations.* CRC Press, Apr. 2016, pp. 1–564.

[Hul+05] R. Hulsebosch, A. Salden, M. Bargh, P. Ebben, and J. Reitsma. "Context sensitive access control". In: *10th ACM Symposium on Access Control Models and Technologies.* Jan. 2005, pp. 111–119.

[JBS15] M. Jones, J. Bradley, and N. Sakimura. *JSON Web Token (JWT).* RFC 7519 (Proposed Standard). RFC. Updated by RFC 7797. Fremont, CA, USA: RFC Editor, May 2015. URL: `https://www.rfc-editor.org/rfc/rfc7519.txt`.

[JH12] M. Jones and D. Hardt. *The OAuth 2.0 Authorization Framework: Bearer Token Usage.* RFC 6750 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Oct. 2012. URL: `https://www.rfc-editor.org/rfc/rfc6750.txt`.

[Jon15] M. Jones. *JSON Web Algorithms (JWA).* RFC 7518 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, May 2015. URL: `https://www.rfc-editor.org/rfc/rfc7518.txt`.

[Jos06] S. Josefsson. *The Base16, Base32, and Base64 Data Encodings.* RFC 4648 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Oct. 2006. URL: `https://www.rfc-editor.org/rfc/rfc4648.txt`.

[Kay+20a] A. S. M. Kayes et al. "A Survey of Context-Aware Access Control Mechanisms for Cloud and Fog Networks: Taxonomy and Open Research Issues". In: *Sensors* 20 (Apr. 2020), p. 2464.

[Kay+20b] A. Kayes, W. Rahayu, P. Watters, M. Alazab, T. Dillon, and E. Chang. "Achieving security scalability and flexibility using Fog-Based Context-Aware Access Control". In: *Future Generation Computer Systems* 107 (June 2020), pp. 307–323. ISSN: 0167-739X.

[KBL18a] D. E. Kouicem, A. Bouabdallah, and H. Lakhlef. "Internet of things security: A top-down survey". In: *Computer Networks* 141 (Aug. 2018), pp. 199–221. ISSN: 1389-1286.

[KBL18b] D. E. Kouicem, A. Bouabdallah, and H. Lakhlef. "Internet of things security: A top-down survey". In: *Computer Networks* 141 (Aug. 2018), pp. 199–221. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2018.03.012. URL: http://www.sciencedirect.com/science/article/pii/S1389128618301208.

[KC03] J. O. Kephart and D. M. Chess. "The vision of autonomic computing". In: *Computer* 36.1 (Jan. 2003), pp. 41–50. ISSN: 1558-0814.

[KCT16] K. Kai, W. Cong, and L. Tao. "Fog computing for vehicular Ad-hoc networks: paradigms, scenarios, and issues". In: *The Journal of China Universities of Posts and Telecommunications* 23.2 (Apr. 2016), pp. 56–96. ISSN: 1005-8885.

[Kim+16] H.-W. Kim, M. R. Hoque, H. Seo, and S.-H. Yang. "Development of Middleware Architecture to Realize Context-Aware Service in Smart Home Environment". In: *Computer Science and Information Systems* 13 (June 2016), pp. 427–452.

[KKV12] E. Kim, D. Kaspar, and J. Vasseur. *Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs).* RFC 6568 (Informational). RFC. Fremont, CA, USA: RFC Editor, Apr. 2012. URL: https://www.rfc-editor.org/rfc/rfc6568.txt.

[KM03] P. Korpipää and J. Mäntyjärvi. "An Ontology for Mobile Device Sensor-Based Context Awareness". In: *Modeling and Using Context, 4th International and Interdisciplinary Conference.* Vol. 2680. June 2003, pp. 451–458.

[Koo11] R. Koodli. *Mobile Networks Considerations for IPv6 Deployment.* RFC 6342 (Informational). RFC. Fremont, CA, USA: RFC Editor, Aug. 2011. URL: https://www.rfc-editor.org/rfc/rfc6342.txt.

[Kot+13] T. Kothmayr, C. Schmitt, W. Hu, M. Bruenig, and G. Carle. "DTLS based Security and Two-Way Authentication for the Internet of Things". In: *Ad Hoc Networks* 11 (Nov. 2013), pp. 2710–2723. ISSN: 1570-8705.

[KPS02] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World.* 2nd. Prentice Hall, Apr. 2002. ISBN: 9780131019904.

[Laa+00] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence. *Generic AAA Architecture.* RFC 2903 (Experimental). RFC. Fremont, CA, USA: RFC Editor, Aug. 2000. URL: https://www.rfc-editor.org/rfc/rfc2903.txt.

[LC16] K.-Y. Lam and C.-H. Chi. "Identity in the Internet-of-Things (IoT): New Challenges and Opportunities". In: *Information and Communications Security.* Ed. by K.-Y. Lam, C.-H. Chi, and S. Qing. Cham: Springer International Publishing, Nov. 2016, pp. 18–26.

[LMS05] P. Leach, M. Mealling, and R. Salz. *A Universally Unique IDentifier (UUID) URN Namespace.* RFC 4122 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, July 2005. URL: https://www.rfc-editor.org/rfc/rfc4122.txt.

[LSA20] S. Latvala, M. Sethi, and T. Aura. "Evaluation of Out-of-Band Channels for IoT Security". In: *SN Computer Science* 1 (Jan. 2020). ISSN: 2661-8907. DOI: 10.1007/s42979-019-0018-8.

[Mah+10a] P. Mahalle, S. Babar, N. R. Prasad, and R. Prasad. "Identity Management Framework towards Internet of Things (IoT): Roadmap and Key Challenges". In: *Recent Trends in Network Security and Applications.* 2010, pp. 430–439.

[Mah+10b] P. Mahalle, S. Babar, N. R. Prasad, and R. Prasad. "Identity Management Framework towards Internet of Things (IoT): Roadmap and Key Challenges". In: *Recent Trends in Network Security and Applications - CNSA 2010.* July 2010, pp. 430–439. ISBN: 978-3-642-14478-3.

[McK+17] K. A. McKay, L. E. Bassham, M. S. Turan, and N. Mouha. *Report on Lightweight Cryptography*. Tech. rep. National Institute of Standards and Technology, Mar. 2017.

[MLA10] D. Martín, C. Lamsfus, and A. Alzua. "Automatic context data life cycle management framework". In: *5th International Conference on Pervasive Computing and Applications*. Dec. 2010, pp. 330–335.

[Moc87] P. Mockapetris. *Domain names - implementation and specification*. RFC 1035 (Internet Standard). RFC. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604, 7766, 8482, 8490. Fremont, CA, USA: RFC Editor, Nov. 1987. URL: https://www.rfc-editor.org/rfc/rfc1035.txt.

[Mon+07] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. RFC 4944 (Proposed Standard). RFC. Updated by RFCs 6282, 6775, 8025, 8066. Fremont, CA, USA: RFC Editor, Sept. 2007. URL: https://www.rfc-editor.org/rfc/rfc4944.txt.

[Mor+04] R. Morgan, S. Cantor, S. Carmody, W. Hoehn, and K. Klingenstein. "Federated Security: The Shibboleth Approach". In: *EDUCAUSE Quarterly* 27 (Jan. 2004).

[MPP13] P. Mahalle, N. R. Prasad, and R. Prasad. "Object Classification based Context Management for Identity Management in Internet of Things". In: *International Journal of Computer Applications* 63 (Feb. 2013), pp. 1–6.

[MV10] R. Maes and I. Verbauwhede. "Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions". In: ed. by R. Maes and I. Verbauwhede. Springer, Berlin, Heidelberg, Oct. 2010, pp. 3–37. ISBN: 978-3-642-14452-3.

[Nah+18] R. K. Naha et al. "Fog Computing: Survey of Trends, Architectures, Requirements, and Research Directions". In: *IEEE Access* 6 (Aug. 2018), pp. 47980–48009. ISSN: 2169-3536.

[Neu+05] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. *The Kerberos Network Authentication Service (V5).* RFC 4120 (Proposed Standard). RFC. Updated by RFCs 4537, 5021, 5896, 6111, 6112, 6113, 6649, 6806, 7751, 8062, 8129, 8429, 8553. Fremont, CA, USA: RFC Editor, July 2005. URL: https://www.rfc-editor.org/rfc/rfc4120.txt.

[NF] P. Nurmi and P. Floréen. *Reasoning in Context-Aware Systems.*

[NJP18] Y. Nir, S. Josefsson, and M. Pegourie-Gonnard. *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier.* RFC 8422 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Aug. 2018. URL: https://www.rfc-editor.org/rfc/rfc8422.txt.

[NLO15] K. T. Nguyen, M. Laurent, and N. Oualha. “Survey on secure communication protocols for the Internet of Things”. In: *Ad Hoc Networks* 32 (Sept. 2015), pp. 17–31. ISSN: 1570-8705.

[NN06] M. Nakhjiri and M. Nakhjiri. “The 3 “A”s: Authentication, Authorization, Accounting”. In: *AAA and Network Security for Mobile Access.* John Wiley and Sons, Ltd, 2006. Chap. 1, pp. 1–23. ISBN: 9780470017463.

[oas22] oasis-open. *XACML Specifications.* July 2017 (accessed May 2, 2022). URL: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[OO16] F. Y. Okay and S. Ozdemir. “A fog computing based smart grid model”. In: *2016 International Symposium on Networks, Computers and Communications (ISNCC).* May 2016, pp. 1–6.

[Oua+17] A. Ouaddah, H. Mousannif, A. A. Elkalam, and A. A. Ouahman. “Access control in the Internet of Things: Big challenges and new opportunities”. In: *Computer Networks* 112 (Jan. 2017), pp. 237–262. ISSN: 1389-1286.

[PB03] P. Prekop and M. Burnett. “Activities, Context and Ubiquitous Computing”. In: *Computer Communications* 26.11 (July 2003), pp. 1168–1176. ISSN: 0140-3664.

[Per+14a] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. "Context Aware Computing for The Internet of Things: A Survey". In: *IEEE Communications Surveys Tutorials* 16.1 (May 2014), pp. 414–454. ISSN: 1553-877X.

[Per+14b] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. "Sensing as a Service Model for Smart Cities Supported by Internet of Things". In: *European Transactions on Telecommunications* (Jan. 2014).

[Por+16] P. Porambage, C. Schmitt, P. Kumar, A. Gurtov, M. Ylianttila, and A. Vasilakos. "The Quest for Privacy in the Internet of Things". In: *IEEE Cloud Computing* 3.2 (Mar. 2016), pp. 36–45. ISSN: 2372-2568.

[Por13] T. Pornin. *Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA).* RFC 6979 (Informational). RFC. Fremont, CA, USA: RFC Editor, Aug. 2013. URL: https://www.rfc-editor.org/rfc/rfc6979.txt.

[PRL09] M. Perttunen, J. Riekki, and O. Lassila. "Context Representation and Reasoning in Pervasive Computing: A Review". In: *International Journal of Multimedia and Ubiquitous Engineering* 4 (Nov. 2009).

[Pul+19] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana. "Fog Computing for the Internet of Things: A Survey". In: *ACM Trans. Internet Technol.* 19.2 (Apr. 2019), 18:1–18:41. ISSN: 1533-5399.

[Rah+14] M. R. Rahimi, J. Ren, C. H. Liu, A. V. Vasilakos, and N. Venkatasubramanian. "Mobile Cloud Computing: A Survey, State of Art and Future Directions". In: *Mobile Networks and Applications* 19.2 (Nov. 2014), pp. 133–143. ISSN: 1572-8153.

[Ram+07] F. Ramparany, R. Poortinga, M. Stikic, J. Schmalenstroer, and T. Prante. "An open context information management infrastructure the IST-amigo project". In: *3rd IET International Conference on Intelligent Environments.* Sept. 2007, pp. 398–403.

[Res18] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3.* RFC 8446 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Aug. 2018. URL: https://www.rfc-editor.org/rfc/rfc8446.txt.

[RG18] P. Rosenberger and D. Gerhard. “Context-awareness in industrial applications: definition, classification and use case”. In: *Procedia CIRP* 72 (June 2018). 51st CIRP Conference on Manufacturing Systems, pp. 1172–1177. ISSN: 2212-8271.

[RLM18] R. Roman, J. Lopez, and M. Mambo. “Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges”. In: *Future Generation Computer Systems* 78 (Jan. 2018), pp. 680–698. ISSN: 0167-739X.

[RM12] E. Rescorla and N. Modadugu. *Datagram Transport Layer Security Version 1.2.* RFC 6347 (Proposed Standard). RFC. Updated by RFCs 7507, 7905. Fremont, CA, USA: RFC Editor, Jan. 2012. URL: https://www.rfc-editor.org/rfc/rfc6347.txt.

[RPM99] N. S. Ryan, J. Pascoe, and D. R. Morse. “Enhanced Reality Fieldwork: the Context-aware Archaeological Assistant”. In: *Computer Applications and Quantitative Methods in Archaeology.* Oct. 1999.

[RR12] V. Radha and D. H. Reddy. “A Survey on Single Sign-On Techniques”. In: *Procedia Technology* 4 (Feb. 2012), pp. 134–139. ISSN: 2212-0173.

[RXA04] M. Rabinovich, Z. Xiao, and A. Aggarwal. “Computing on the Edge: A Platform for Replicating Internet Applications”. In: *Web Content Caching and Distribution.* Ed. by F. Douglis and B. D. Davison. Springer Netherlands, 2004, pp. 57–77.

[RZL13] R. Roman, J. Zhou, and J. Lopez. “On the features and challenges of security and privacy in distributed internet of things”. In: *Computer Networks* 57 (July 2013), pp. 2266–2279. ISSN: 1389-1286. URL: http://www.sciencedirect.com/science/article/pii/S1389128613000054.

[Sal+16a] J. Saldana (Ed.), A. Arcia-Moret, B. Braem, E. Pietrosemoli, A. Sathiaseelan, and M. Zennaro. *Alternative Network Deployments: Taxonomy, Characterization, Technologies, and Architectures.* RFC 7962 (Informational). RFC. Fremont, CA, USA: RFC Editor, Aug. 2016. URL: https://www.rfc-editor.org/rfc/rfc7962.txt.

[Sal+16b] O. Salman, S. Abdallah, I. Elhajj, A. Chehab, and A. Kayssi. "Identity-based authentication scheme for the Internet of Things". In: *2016 IEEE Symposium on Computers and Communication (ISCC).* June 2016, pp. 1109–1111.

[San+06] L. Sanchez, J. Lanza, R. Olsen, M. Bauer, and M. Girod-Genet. "A Generic Context Management Framework for Personal Networking Environments". In: *3rd Annual International Conference on Mobile and Ubiquitous Systems - Workshops.* July 2006, pp. 1–8.

[San+13] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP.* RFC 6960 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, June 2013. URL: https://www.rfc-editor.org/rfc/rfc6960.txt.

[San93] R. S. Sandhu. "Lattice-based access control models". In: *Computer* 26 (Nov. 1993), pp. 9–19. ISSN: 1558-0814.

[San98] R. S. Sandhu. "Role-based Access Control". In: ed. by M. V. Zelkowitz. Vol. 46. Advances in Computers. Elsevier, 1998, pp. 237–286.

[Sar+15] C. Sarkar, A. U. Nambi S. N., R. V. Prasad, A. Rahim, R. Neisse, and G. Baldini. "DIAT: A Scalable Distributed Architecture for IoT". In: *IEEE Internet of Things Journal* 2 (June 2015), pp. 230–239. ISSN: 2372-2541.

[Sat+09] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. "The Case for VM-Based Cloudlets in Mobile Computing". In: *IEEE Pervasive Computing* 8.4 (Oct. 2009), pp. 14–23. ISSN: 1558-2590.

[Sat+13] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha. “The Role of Cloudlets in Hostile Environments”. In: *IEEE Pervasive Computing* 12.4 (Oct. 2013), pp. 40–49. ISSN: 1558-2590.

[Sat15] M. Satyanarayanan. “A Brief History of Cloud Offload: A Personal Journey from Odyssey Through Cyber Foraging to Cloudlets”. In: *GetMobile: Mobile Comp. and Comm.* 18.4 (Jan. 2015), pp. 19–23. ISSN: 2375-0529.

[SAW94] B. Schilit, N. Adams, and R. Want. “Context-Aware Computing Applications”. In: *First Workshop on Mobile Computing Systems and Applications*. Dec. 1994, pp. 85–90.

[SBF98] R. Studer, V. Benjamins, and D. Fensel. “Knowledge engineering: Principles and methods”. In: *Data & Knowledge Engineering* 25.1 (1998), pp. 161–197. ISSN: 0169-023X.

[Sch+17] P. Schulz et al. “Latency Critical IoT Applications in 5G: Perspective on the Design of Radio Interface and Network Architecture”. In: *IEEE Communications Magazine* 55.2 (Feb. 2017), pp. 70–78. ISSN: 1558-1896.

[Sha+17] S. Shahzadi, M. Iqbal, Z. Qayyum, and T. Dagiuklas. “Infrastructure as a Service (IaaS): A Comparative Performance Analysis of Open-Source Cloud Platforms”. In: *2017 IEEE 22nd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. June 2017.

[Sha+18] K. Sha, W. Wei, T. [Yang], Z. Wang, and W. Shi. “On security challenges and open issues in Internet of Things”. In: *Future Generation Computer Systems* 83 (June 2018), pp. 326–337. ISSN: 0167-739X. URL: http://www.sciencedirect.com/science/article/pii/S0167739X17324883.

[Shi+16] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. “Edge Computing: Vision and Challenges”. In: *IEEE Internet of Things Journal* 3.5 (Oct. 2016), pp. 637–646. ISSN: 2372-2541.

[Shi07] R. Shirey. *Internet Security Glossary, Version 2.* RFC 4949 (Informational). RFC. Fremont, CA, USA: RFC Editor, Aug. 2007. URL: https://www.rfc-editor.org/rfc/rfc4949.txt.

[SL04] T. Strang and C. Linnhoff-Popien. “A Context Modeling Survey”. In: *Proceedings of the Workshop on Advanced Context Modeling, Reasoning and Management as Part of UbiComp.* Sept. 2004.

[Sot+09] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. “Virtual Infrastructure Management in Private and Hybrid Clouds”. In: *IEEE Internet Computing* 13 (Sept. 2009), pp. 14–22. ISSN: 1941-0131.

[SSD09] F. Skopik, D. Schall, and S. Dustdar. “Start Trusting Strangers? Bootstrapping and Prediction of Trust”. In: *Web Information Systems Engineering - WISE 2009.* 2009, pp. 275–289.

[Sta11] M. Stamp. *Information Security: Principles and Practice.* 2nd. Wiley Publishing, May 2011. ISBN: 0470626399.

[Sta16] W. Stallings. *Network Security Essentials: Applications and Standards.* 6th. Pearson, 2016. ISBN: 013452733X.

[SU22] I. Strategy and P. Unit. *ITU Internet Reports 2005: The Internet of Things.* Tech. rep. Geneva: International Telecommunications Union(ITU), Nov. 2005 (accessed May 2, 2022). URL: https://www.itu.int/osg/spu/publications/internetofthings/.

[TC16] M. Trnka and T. Cerny. “Identity Management of Devices in Internet of Things Environment”. In: *2016 6th International Conference on IT Convergence and Security (ICITCS).* Sept. 2016, pp. 1–4.

[TDM10] L. Toka, M. Dell’Amico, and P. Michiardi. “Online Data Backup: A Peer-Assisted Approach”. In: *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P).* Aug. 2010, pp. 1–10.

[Ted+20] P. Tedeschi, S. Sciancalepore, A. Eliyan, and R. Di Pietro. "LiKe: Lightweight Certificateless Key Agreement for Secure IoT Communications". In: *IEEE Internet of Things Journal* 7 (Nov. 2020), pp. 621–638. ISSN: 2327-4662.

[TTČ17] M. Trnka, M. Tomasek, and T. Černý. "Context-Aware Security Using Internet of Things Devices". In: *ICISA: Information Science and Applications*. Mar. 2017, pp. 706–713.

[Val+16] C. Vallati, A. Virdis, E. Mingozzi, and G. Stea. "Mobile-Edge Computing Come Home Connecting things in future smart homes using LTE device-to-device communications". In: *IEEE Consumer Electronics Magazine* 5.4 (Oct. 2016), pp. 77–83. ISSN: 2162-2256.

[VZN12] H. Vahdat Nejad, K. Zamanifar, and N. Nematbakhsh. "Context-Aware Middleware Architecture for Smart Home Environment". In: *International Journal of Smart Home* 7.1 (Jan. 2012).

[Wan+04] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. "Ontology based context modeling and reasoning using OWL". In: *IEEE Annual Conference on Pervasive Computing and Communications Workshops*. Mar. 2004, pp. 18–22.

[Wan+14] J. Wan, D. Zhang, S. Zhao, L. T. Yang, and J. Lloret. "Context-aware vehicular cyber-physical systems with cloud support: architecture, challenges, and solutions". In: *IEEE Communications Magazine* 52.8 (Aug. 2014), pp. 106–113. ISSN: 0163-6804.

[YLL15] S. Yi, C. Li, and Q. Li. "A Survey of Fog Computing: Concepts, Applications and Issues". In: *Proceedings of the 2015 Workshop on Mobile Big Data*. Mobidata '15. Association for Computing Machinery, 2015, pp. 37–42.

[Zho+17] J. Zhou, Z. Cao, X. Dong, and A. V. Vasilakos. "Security and Privacy for Cloud-Based IoT: Challenges". In: *IEEE Communications Magazine* 55.1 (Jan. 2017), pp. 26–33. ISSN: 1558-1896.

[Zhu+17] X. Zhu, Y. Badr, J. Pacheco, and S. Hariri. "Autonomic Identity Framework for the Internet of Things". In: *2017 International Conference on Cloud and Autonomic Computing (ICCAC)*. Sept. 2017, pp. 69–79.