J. Umm Al-Qura Univ. Eng.Archit.
https://doi.org/10.1007/s43995-025-00149-6

ORIGINAL ARTICLE

Containerized attribute-based access control system using digital keys

Samer I. Mohamed[1] · **Manal Mostafa**[2] · **Jalal Assaly**[1] · **Ahmed S. Shalabi**[1]

Received: 24 September 2024 / Accepted: 15 May 2025

Abstract

The Containerized Attribute-Based Access Control System (ABACS) using Digital Keys provides an efficient means of granting or revoking access to users in residential and commercial buildings. Majorly used credential technologies show an absence of encryption capabilities, performance challenges and present a lack of proper scalability. The proposed system, ABACS, offers a container-based access control solution with enhanced security, scalability and performance, via a user-friendly management, and a convenient mobile application. ABACS system Authentication, integrity, and confidentiality are guaranteed using multiple security methods, including a Trusted Execution Environment (TEE) for safe digital key encryption, and the Transport Layer Security (TLS) protocol for secure channel communication, supported by a digital certificate. Performance is achieved through the use of the Constrained Application Protocol (CoAP) for embedded system internet communication, and the Near-Field Communication (NFC) channel for quick digital key sharing. Access control and user management is achieved using the Attribute-Based Access Control (ABAC) model deployed on-premise. ABACS applies the principles of containerization to enable modularity, service isolation, and horizontal scalability, critical features for supporting large-scale system distribution. ABACS effectively mitigates major attack vectors, including man-in-the-middle, replay (both internet and NFC), credential cloning, and unauthorized mobile access through a combination of per-session nonces, TLS/DTLS-secured channels, tamper-aware embedded controllers, and backend-enforced policies. These layered protections offer stronger guarantees compared to prior systems, many of which overlook or partially address such threats. Performance evaluations confirm ABACS's backend is both scalable and responsive. In sequential request handling, ABACS processes requests at least ten times faster than iPACS. Under concurrent load, it maintains more than double the throughput, demonstrating robust system efficiency and supporting real-world multi-user environments. In terms of user-friendliness, ABACS delivers a streamlined and intuitive mobile experience. Users register and authenticate with minimal effort through biometric and login credentials, avoiding the friction of manual security code entry or reliance on physical Radio-Frequency Identification (RFID) cards. This modern design improves usability and adoption while maintaining strong security guarantees.

Keywords Attribute-based access control · Digital keys · Trusted layer security · Near-field communication · Containerization

1 Introduction

The rapid evolution of the Internet of Things (IoT) has made it a crucial technology in today's digital world. IoT systems simplify our daily activities, enhance systems security, and offer great user convenience. One of the areas that witnessed a large development thanks to IoT technologies is access control systems in smart buildings. A Physical Access Control System (PACS) provides the ability of limiting entrance privileges to a room, a building, or a property to authorized users only [1]. The traditional access method remained for long a simple key and keylock system. Whoever possessed the key had the access rights to enter the restricted area.

✉ Samer I. Mohamed
dr.samer.ibrahim@gmail.com; saibrahim@msa.edu.eg

Manal Mostafa
manal.mustafa@azhar.edu.eg

Jalal Assaly
jalal.assaly@msa.edu.eg

Ahmed S. Shalabi
ahmed.salah25@msa.edu.eg

[1] Computer Systems Engineering, October University for Modern Sciences and Arts, Giza, Egypt

[2] Computer and Systems Engineering, Al-Azhar University, Cairo, Egypt

Published online: 28 May 2025

However, as time passed and issues were identified, new credential technologies and methods of access validation emerged. The most noticeable changes in the industry happened in the last 10–15 years. Multiple sorts of user credentials were developed and implemented, such as Quick Response (QR) codes, RFID, biometric and mobile-based technologies, just to name a few.

Over the years, the industry's interests and focus shifted from one concern to the other. For instance, with the advent of IoT and network-connected devices, cybersecurity became one of the top challenges faced by companies. According to a worldwide survey conducted in collaboration between IFSEC and HID Global, a trusted provider of secure identity and access control solutions, 40% of the 1000 respondents cited "protecting against the threat of security vulnerabilities as a top challenge" [2]. Another primary challenge faced by 37% of the respondents to the same survey, is being capable of issuing and revoking ID credentials efficiently. In fact, managing credentials efficiently stems from a more general concern that is to provide user convenience. Around 43% of the respondents claimed that their major concern was to make the management of physical access control easier. Other noteworthy contemporary challenges are the ability to integrate physical access control with other infrastructure systems and, to reduce physical touchpoints as cited by 27% and 13% of the respondents respectively.

Overall, the industry of PACSs currently faces important and severe concerns, such as security threat prevention, ease of management and use, integration with open platforms and touchless solutions. PACSs are security solutions that regulate and monitor entry to physical spaces, such as buildings or rooms. In the past two decades, these systems rapidly went from the traditional mechanical keys to the more modern digital keys, often in the form of electronic cards or mobile credentials, to grant or restrict access. Recent advancements in this field include sophisticated authentication methods, such as secure RFID smart cards and biometrics (e.g., fingerprint, retina scans, face ID), and integration with smart technologies and established building management systems. Various local and international competitors contribute to this sector, each with distinct market shares and product features. In 2020 [2], the market for physical access control solutions was worth approximately $5.66 billion and was projected to grow by 7.3% each year to reach $8.07 billion by 2025. However, after the COVID-19 pandemic, the rate of adoption for physical access control systems, aimed to aid in disease prevention and user monitoring, has probably pushed this estimation even higher. As of 2023, the market is still in rapid expansion and new technologies see the light of research and adoption every day.

To efficiently manage PACSs, the proposed system is implemented using an ABAC model, which grants or revokes access based on the concurrent attributes of users (e.g., roles), access points (e.g., server room) and environmental factors (e.g., time). In addition, the system shall focus on creating a seamless user experience using mobile devices. A mobile smartphone solution shall be devised to act as a wallet for the digital keys, and as such, a mobile application should be developed. It aims to comprehensively analyze and implement several security measures, including device, internet, and key generation security. The system will address these security concerns by implementing the latest NFC protocol alongside a TEE in the smartphone's operating system to establish a seamless and fortified physical environment, guaranteeing the secure transmission and storage of the sensitive digital keys. Furthermore, the TLS protocol will be used in both symmetric and asymmetric modes, to ensure robust security at the transport and application layers during internet communication. The TLS asymmetric encryption will be supported by a digital certificate that shall add server authentication features. Additionally, to achieve multi factor authentication, the system relies on multiple authenticating credentials (e.g., fingerprint, login credentials), with the correct combination of authenticating attributes and finally, frequently generated nonces to provide one-time digital keys. The proposed system, ABACS, offers strong protection against common threat models and attacks such as man-in-the-middle, replay, cloning and unauthorized app access. In addition to its security design, ABACS also demonstrates robust scalability and performance. Backend components efficiently handle both sequential and concurrent access requests, outperforming previous systems by a wide margin. This ensures the system remains responsive under high loads, making it suitable for deployment in large-scale smart building environments.

In addition to its security design, ABACS incorporates a modular architecture based on the concept of containerization. This practice involves packaging each service with its required dependencies into lightweight, isolated environments that can run consistently across different infrastructures. Containerization supports improved service isolation, resilience, and resource efficiency while simplifying deployment across a variety of devices. More importantly, this architecture supports horizontal scalability, allowing multiple components of the system to be replicated and distributed efficiently across different nodes in large-scale environments. This scalability model was conceptually validated through performance benchmarking, which demonstrated that the system remains responsive and capable of handling multiple concurrent requests.

Lastly, the system was designed with user-friendliness as a core priority. ABACS eliminates manual credential handling and hardware dependencies by leveraging familiar smartphone workflows. From registration to access, users interact with intuitive biometric and credential-based authentication mechanisms, while the backend handles

validation seamlessly. In contrast to existing systems that rely on external RFID cards or custom hardware, ABACS delivers a modern, frictionless access experience aligned with today's user expectations.

The proposed contribution lies in ABACS's unique integration of secure communication protocols, mobile-first usability, and a physically enforced attribute-based access control model, an approach not jointly implemented in any of the reviewed systems. Unlike prior solutions that rely on static credentials, external secure elements, or hardware-bound identifiers, ABACS introduces per-session digital keys, nonce-based authentication, and multi-factor mobile login in a unified architecture. Its layered security model is coupled with a lightweight, intuitive user workflow, achieving a level of adaptability, protection, and convenience that sets it apart from current state-of-the-art PACS implementations. ABACS is not merely an access control system, it is a comprehensive, scalable, and context-aware solution tailored for modern smart building environments. The rest of this paper is organized as follows: literature review and design of the proposed system are discussed in Sects. 2 and 3 respectively. Evaluation results are presented in Sect. 4 followed by conclusions and issues for future research in Sect. 5.

2 Background and related work

The evolution of physical access control systems has witnessed a transformative journey from traditional lock-and-key mechanisms to sophisticated, technology-driven solutions that define the current state of the art. In the early stages, mechanical locks and physical keys were the primary means of securing access to buildings and rooms. The limitations of these systems, including the risk of lost or duplicated keys, pushed the development of electronic access control systems. The advent of electronic access control systems marked a significant shift, introducing key cards and key fobs that could be easily managed and monitored. These systems provided enhanced security through audit trails, enabling administrators to track access events and manage permissions more efficiently. Magnetic stripe cards, characterized by a black strip on their back, were the first type of cards in use. Then, with the uprising of RFID technologies, newer more convenient cards were conceived. Often termed as proximity cards, the low frequency 125 kHz RFID cards are read from a close distance without the need for physical contact. However, since the first introduction of electronic access control systems in the 1980s and 1990s, magnetic and proximity cards can no longer be termed as major improvements since they lack basic security features such as encryption. Hence, following this constatation, newer electronic RFID cards called smart cards, operating at 13.56 MHz, were designed and implemented, to provide a higher and smarter security access solution. Later, as technology advanced, biometric authentication, such as fingerprint, facial recognition, and retina scans, became integrated into access control solutions, adding an additional layer of security and eliminating the need for physical tokens. The market for biometric-based solutions is on a current uptake where 30% of the HID Global survey participants stated actively using it [2].

In the current state of the art, access control systems leverage the power of the IoT and connectivity. Cloud-based solutions allow for centralized management and real-time monitoring, providing administrators with greater flexibility and control. Furthermore, the software parts of the system need to be more frequently updated than the hardware where cloud-based solutions can easily provide patches and improved performance to the components through regular updates. Mobile credentials, utilizing smartphones as digital keys, have gained popularity, offering convenience and improved user experiences. The market perfectly reflects this observation as 32% of the interviewees cited actively using mobile IDs [2].

The Role-Based Access Control (RBAC) model, comprising users, roles, operations, objects, and permissions, establishes a dynamic framework where permissions are assigned to roles, and users are associated with specific roles. The role-centric approach of RBAC aims to shift the permission control from being assigned to specific users, to being associated with more general engineered roles. This methodology ensures logical independence through role-based policies, adherence to the least privileges principle, and support for the separation of duties. RBAC's versatile application across domains, together with its ability to simplify security policy management and administration through role-based structures, makes it a powerful model for access control in diverse organizational contexts [3].

ABAC is introduced as a model operating with attributes, which are associated with three core entities: subjects, objects, and environment. Each of these entities can hold multiple attributes, which are used by the access policies to make access decisions. Subject attributes contain information about the user requesting access; the subject may include the role, job title, and subject ID. Object attributes could include the maximum number of users in the room. Finally, environmental attributes may encompass factors such as the current time and the emergency status of the environment [3].

Arnosti et al. [4] propose a smartphone-based PACS that authenticates users online via a central server, while remaining independent from third-party providers such as mobile network operators and handset manufacturers. They explore two architectural models: Host Card Emulation (HCE) and a tamper-resistant microSD-based Secure Element (microSD-SE). While HCE offers ease of deployment, it is vulnerable

to malware and software-based relay attacks, as sensitive operations occur in the device's main OS. In contrast, the microSD-SE variant stores credentials securely and performs cryptographic operations independently, communicating directly with the NFC reader and bypassing the OS during authentication. This significantly enhances protection against credential theft and unauthorized access. However, the system still presents practical deployment challenges. As many modern smartphones no longer include microSD card slots, the approach becomes difficult to implement in bring-your-own-device (BYOD) environments. Additionally, the use of external hardware introduces friction in terms of scalability and user adoption. While the microSD-SE model improves upon HCE by offering stronger cryptographic isolation, its dependence on specialized hardware limits its viability in widespread real-world applications.

Petrakis et al. [5] introduce iPACS, a cloud-based physical access control system aimed at managing access and monitoring user activity in large public and residential spaces. The system uses BLE beacons to broadcast area identifiers to nearby smartphones, which interact with a private fog-based cloud to request access and report user movement. The system supports dynamic access rules based on roles or subscriptions and enables emergency alerts and anonymized reporting to a public cloud for data analysis. Despite its modular design, iPACS faces limitations in scalability, responsiveness, and usability. Performance testing highlights high latency under heavy load, due in part to synchronous service communication using CURL and limited resource allocation across virtual machines. Security-wise, while it applies OAuth 2.0 and encrypts data at the application level, the use of Base64 encoding offers no cryptographic protection. These constraints reduce its effectiveness in high-assurance or time-sensitive deployments.

Reza et al. [6] present an IoT-based door access control system that leverages Wi-Fi communication and mobile biometrics to authenticate users and manage access remotely. The system uses a Raspberry Pi as the central controller and a mobile application that performs dual-factor authentication via fingerprint recognition and the device-specific International Mobile Equipment Identity (IMEI) number. During registration, users input their mobile device IMEI on the local door reader using the connected keypad. During login, the IMEI is automatically encrypted and sent to a cloud server, where it is later retrieved and compared to the locally stored copy on the access controller. Access is granted only if both the fingerprint and IMEI match and the mobile device is connected to the local Wi-Fi network. The authors highlight the system's ability to perform sensitive operations, such as encryption and device identification, invisible to the user, aiming to enhance both security and ease of use. Despite these efforts, the system presents notable limitations in terms of scalability and flexibility. Because access is tied to a single registered IMEI, the system cannot easily accommodate users who wish to change or use multiple devices, nor does it support multi-user environments without manual reconfiguration. This tight binding to hardware identifiers also introduces challenges in managing large or dynamic user populations. While biometric authentication adds a layer of security, the overall reliance on static identifiers and local verification mechanisms limits adaptability and introduces potential single points of failure in both the registration and decision processes.

Noprianto et al. [7] proposes an RFID-based door access control system that adopts a microservices architecture and utilizes the Message Queuing Telemetry Transport (MQTT) protocol to streamline communication between access points and backend services. The system includes modular components for access validation, log collection, and device and personnel management through a web-based frontend and backend built using PHP and MySQL. Communication is optimized by encapsulating low-level socket operations through MQTT and Hypertext Transfer Protocol (HTTP) web services. Authentication is handled using JWT tokens with SHA-256 hashing, while RFID card-level security is enforced through Key A and Key B diversification, enabling sector-based control and local verification. Despite its modularity and platform independence, the system presents several limitations. The use of static RFID credentials restricts its resilience against cloning and replay attacks, especially in the absence of dynamic token validation or session-based access. Scalability may also be challenged as MQTT performance under high-concurrency, real-time access scenarios is not evaluated. Usability is further constrained by the need for physical RFID cards which limit support for multi-factor authentication models.

3 ABACS architecture and design

The ABACS system is composed of three core components. First, an embedded access point device is installed at restricted entries to detect and forward user access attempts. Second, a local server processes these requests, enforces access policies, and monitors overall system activity. Third, a mobile application serves as the user's secure digital key, allowing them to authenticate and initiate access via NFC. Together, these layers form a modular and scalable architecture. The following sections detail each component and the design considerations behind their integration.

3.1 Technical specifications

The proposed ABACS system involves the development and implementation of a comprehensive access control solution utilizing various technical specifications and components.

The system is centered around a downloadable Android-based mobile application that interfaces with an NFC-enabled mobile device, enabling communication of digital keys to a PN532 NFC reader via a 13.56 MHz radio frequency. Access control is managed through a 12 V Mini Solenoid Door Lock installed on an MDF Wooden Door with a frame, operated by a 5 V DC relay module to control the locking and unlocking mechanism. User interaction and feedback are provided through an integrated LCD, LED indicators, and a Buzzer, offering real-time updates on the access status.

At the core of the system's operation is an ESP32 microcontroller, which utilizes its WiFi capability to orchestrate the interactions between the components. Additionally, a Raspberry Pi model 4B functions as a dedicated local server, optimizing response times by processing access requests locally. Security is reinforced through the application of an ABAC model, enabling dynamic access policy generation and management on the backend server. Communication between the access point and backend server is secured using HTTP application and TLS encryption protocols for both intranet and internet communications, in addition to CoAP and Datagram TLS (DTLS) protocols to ensure efficient and secure access request authorization. This integrated approach guarantees both operational efficiency and high security in managing access control scenarios, making it suitable for deployment in various environments that require stringent access management protocols.

3.2 ABAC model

ABAC is an access control model that essentially compares a set of predefined rules called access policies, with a set of descriptive information called attributes from various system objects and entities. The attributes are validated by the access policies in order to produce an access decision.

3.2.1 Attributes

The ABAC has 3 main arbitrary attributes which are subject attributes, resource attributes and environment attributes. Each of these attributes plays a role in the ABAC to give the Access Control (AC) to the right subject. A subject, whether it's a user, application, or process, is an entity that interacts with a resource. Each subject has its own specific attributes like (User ID, Role, Department, Time schedule, Clearance level, User status) that establish their identity and characteristics. A resource, which could be something like a web service, a physical or system component, is an entity that undergoes actions performed by a subject. Much like subjects, resources also contain attributes like (Access Point ID, Location, Tamper detection, Occupancy level) that can be utilized to inform AC decisions. Environmental attributes like (Date, Time, Emergency status) characterize the operational, technical, and contextual aspects of the environment in which information access takes place.

3.2.2 Policy model

The ABAC policy model is characterized by, S, R and E which are respectively the Subject, Resource and Environment. Moreover, the attributes that have been established in advance for subjects, resources, and environments are going to be represented with the following equations respectively [8].

$$SAx\ (1 \le x \le X), RAy\ (1 \le y \le Y), \text{ and } EAz\ (1 \le z \le Z). \tag{1}$$

Furthermore, the attribute assignment relations for the subject, resource and environment are going to be represented respectively as follows ATTR(s), ATTR(r), and ATTR(e),

$$ATTR(s) \subseteq SA1 \times SA2 \times \cdots \times SAx \tag{2}$$

$$ATTR(r) \subseteq RA1 \times RA2 \times \cdots \times RAy \tag{3}$$

$$ATTR(e) \subseteq EA1 \times EA2 \times \cdots \times EAz, \tag{4}$$

These relations specify how attributes are assigned to subjects, resources, and environments. The Policy Rule (PR) is responsible for determining whether a subject (s) can have an AC on a resource (r) in the environment (e). The PR in the most general form can be as follows, can_access $(s, r, e) \leftarrow f(\text{ATTR}(s), \text{ATTR}(r), \text{ATTR}(e))$. Considering all the attributes assigned to each one of them, the function evaluates and provides an output. If the output is True, AC is granted; otherwise, AC is denied [8].

The Attribute Authorities (AA) administrators are the ones who are responsible for maintaining (creating, managing, etc.) the attributes for subjects, resources, and environments. This means they are responsible for the characteristics or properties that describe subjects, resources, and the environment. AA may or may not store the attributes by themselves, but in any case, they are responsible for associating or "binding" attributes to the entities they describe. The Policy Authority (PA) administrators are responsible for formulating and overseeing access control policies. These can include decision rules, conditions, and additional constraints related to resource access. The Policy Administration Point (PAP) is the software component used by the AA and PA to design and define the access policies, rules and attributes that will be used to grant or revoke access. It handles common functions such as create, manage, test, and debug access attributes and policies, and to store the attributes definitions and policies in the designated database. The Policy Enforcement Point (PEP) is the software component

responsible for receiving access requests, demanding authorization decisions, and enforcing them. Essentially, it serves as the AC point and should have the capability to intercept service requests between those seeking information and those supplying it [9]. The Policy Decision Point (PDP) is the core software component of the system, and it plays an important role which is evaluating the policies and attributes against the rules provided by the PA and AA, respectively, to generate an access control decision. Furthermore, it is tasked with assessing the relevant policies and rendering an authorization decision (grant or deny). Essentially, the PDP functions as a policy execution engine. If a policy refers to a subject, resource, or environmental attribute that is absent from the request (e.g., department missing), the PDP communicates with the PEP to trigger an alerting mechanism. The information is then relayed to the PA and AA administrators who will adjust the access policies and rules where necessary to solve the issue. The Policy Information Point (PIP) is a software component responsible for fetching the additional attributes necessary for policy evaluation, such as the environment attributes for example. Its role is to supply the essential information to the PDP to facilitate its decision-making [10].

3.3 Transport layer security (TLS)

In the proposed ABACS, the mobile device as well as the embedded access control system need to communicate with the local server over the internet; hence, there is the need to use an adequate internet protocol that provides privacy, data security and reliability. The TLS protocol has been chosen as one of the most widely adopted methods that satisfies the challenging security concerns and is nowadays a critical component for global trusted internet communication [11]. TLS can be decomposed into four main phases: key exchange, authentication, bulk encryption, and hashing. The four algorithms are often described in a cipher suite which details the algorithms being used for each phase. For example, a cipher suite can be "ECDHE_ECDSA_WITH_AES128_SHA256", meaning that it utilizes Elliptic-Curve Diffie-Hellman Ephemeral (ECDHE) for key exchange, Elliptic-Curve Digital Signature Algorithm (ECDSA) for authentication, Advanced Encryption Standard 128-bits (AES128) for bulk encryption and Secure Hash Algorithm 256-bits (SHA256). Depending on the used cipher suite, the protocol's confidentiality, integrity, and responsiveness is impacted [11]. The most recent TLS 1.3 version was selected as it provides many optimized features and satisfies the important computer security pillars of confidentiality, integrity, availability, authenticity, non-repudiation and more. However, the most important feature provided by the latest TLS protocol is the perfect forward secrecy. It consists of providing enough security to the exchange so that at any point in the future, if the secured keys were to ever be compromised, any data that was previously sent can never be decrypted. A CA will be used to generate digital certificates to the backend server to enhance the integrity and authentication of the system components.

3.4 Constrained application protocol (CoAP)

To establish communication between devices over the internet, application layer protocols are used to govern the shape, size and method of requests and responses. HTTP is the most convenient application protocol used for data communication over traditional networks, containing capable devices with no limitations on resources or power consumption. However, when it comes to IoT devices, HTTP has shown to be a particularly wasteful protocol that does not comply with the constrained nature of the network devices and bandwidth and power constraints available. Hence, newer protocols were developed to solve this issue. CoAP is an application layer protocol specifically developed for constrained environments and devices commonly found in IoT networks. It guarantees low processing overhead, optimized power consumptions, and fast communication bandwidth when compared to HTTP. CoAP is designed to resemble HTTP. It operates on a REST-style architecture, uses Uniform Resource Identifiers (URIs) to identify network resources and services, and initiates requests through the GET, POST, PUT and DELETE methods. By default, HTTP works over the Transmission Control Protocol (TCP) and CoAP over User Datagram Protocol (UDP) [12]. For these different reasons CoAP was adopted in the proposed system.

3.5 Near-field communications (NFC)

NFC technology, embedded within most mobile smartphones nowadays. First, it provides high-portability, user convenience, and it ensures that users will not forget or lose their credentials, as carrying mobiles around has become a standard habit for everyone. Secondly, NFC is a cost-effective solution, because it does away with physical cards which entails a high cost of production and maintenance. Finally, it grants a reliable and secure way of communication due to its operating radio frequency being established at 13.56 MHz, which enables fast and encrypted-capable communication [13]. Therefore, NFC will be used as a physical communication protocol to provide a secure mobile access solution.

3.6 Trusted execution environment (TEE)

A TEE is a secure and isolated section within the primary processor (CPU) of devices, such as smartphones. Its primary purpose is to ensure the secure storage, processing, and protection of sensitive data in a trusted and isolated

environment [14]. The processor controls a dual operating system (OS) environment respectively called the Rich OS and the Trusted OS. The Rich OS serves as the device's primary operating system. It controls and manages general functions and user applications, essentially acting as an interface for everyday interactions. In parallel, the Trusted OS operates within the TEE. It is responsible for critical security functions and the protection of sensitive data. Within this isolated execution environment, the TEE fulfils a range of important security requirements. Its security architecture is characterized by its ability to enable isolated execution, provide secure storage for both binaries and application data, support remote attestation for authenticity verification, facilitate secure provisioning of data, and establish a trusted communication path with the external world. These collective measures fortify the TEE against a variety of security threats, and satisfy the principles of code authentication, confidentiality, authenticity, privacy, system integrity, and precise control over data access rights, which emphasizes its importance as one of the core pillars of the proposed system [14].

3.7 Mobile application

As concluded from the current state-of-the art, the future trends as well as the literature review, mobile-based solutions will dominate the future of PACSs. Therefore, the objective is to build a user-friendly mobile application for ABACS to generate and store digital keys. The digital keys are generated using the user attributes and encrypted using the public key of the local server. The mobile application will also implement a local authentication process such as entering a Personal Identification Number (PIN) code or scanning a fingerprint to provide an added factor of authentication. Coupled with the server authentication for the digital key, the system can therefore achieve a multifactor authentication mechanism. The digital key will be encrypted using AES128 and generated with a nonce that is added as a One-Time Password (OTP) and ensures the liveness of the session.

3.8 ABACS architecture

3.8.1 Block diagram

The block diagram shown in Fig. 1 provides a high-level overview of the proposed ABACS components and modules, along with the logic connecting these entities. Each block in the diagram represents a hardware or software component or subsystem, assisting in the identification of essential elements within the system. The system includes several key components. The *User* refers to an employee or a visitor who seeks access to the restricted area. The *Smartphone* is a smart device equipped with NFC capabilities, responsible

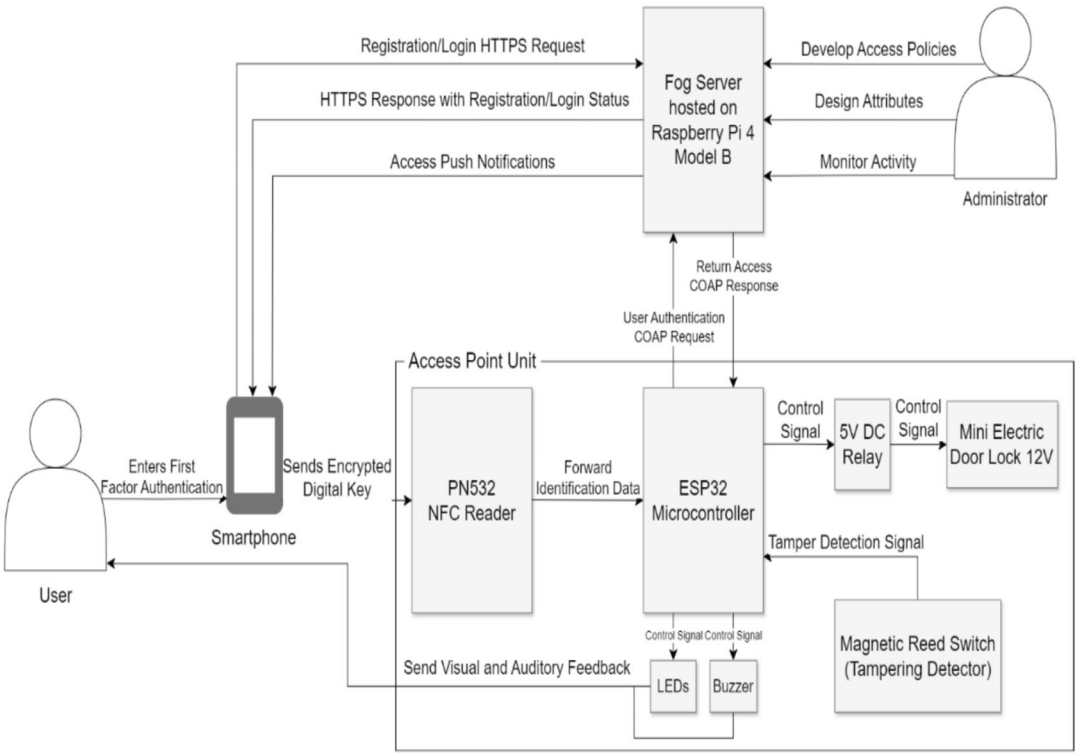


Fig. 1 Core APACS components block diagram

for generating, storing, and transmitting the user's digital key to the access point reader. Through the mobile application, the smartphone presents an authentication portal to the PACS online system, displaying the user's profile page and access history. The *Access Point Unit* is a collection of electronic components that captures, preprocesses, and communicates the user's digital key to the local server. Additionally, it enforces the access decision on the access point door. The *Local Server* acts as the central hub for communication and control within the PACS. It performs four primary tasks: user login and registration, access control and authentication, access policies and attributes design, and system monitoring. Finally, the *Administrator* is an organization agent responsible for managing, monitoring, and troubleshooting the PACS's activity and performance. The administrator also oversees the creation and management of access policies for the system.

Figure 2 represents the core logic which resides inside the local server. The server is structured according to the ABAC model described previously in Sect. 3.2. In addition to those components, more business logic can be performed by the server. For instance, the user activity can be monitored and saved as logs for security and auditing purposes. Policies are created by an admin through the Policy Administration Point (PAP) and stored in the Policy Repository (PR). When a user sends an access request, the ESP32 device (acting as a Policy Enforcement Point or PEP) forwards the request using the CoAP protocol to the Policy Decision Point (PDP). The PDP consults stored policies and relevant contextual data from the Policy Information Point (PIP) to determine whether access should be granted. The result is sent back to the ESP32 as a CoAP response. Meanwhile, any relevant activity is also forwarded to extra backend logic for further processing and logging, thereby supporting accountability and system monitoring.

3.8.2 Flowchart diagram

The flowchart summarizes in great lines the events and steps achieved during the normal operation of the system. Decisions are also reflected along the diagram to show the different branches and outcomes the system undergoes. Figure 3 shows the flowchart of the proposed ABACS, along with the important activities, decision branches and database stores.

1. Firstly, at the very beginning of the system flow, users open their mobile device and are presented with login or registration options to access the ABACS. If users are not already registered, they are prompted to register with their full user credentials. Otherwise, they can

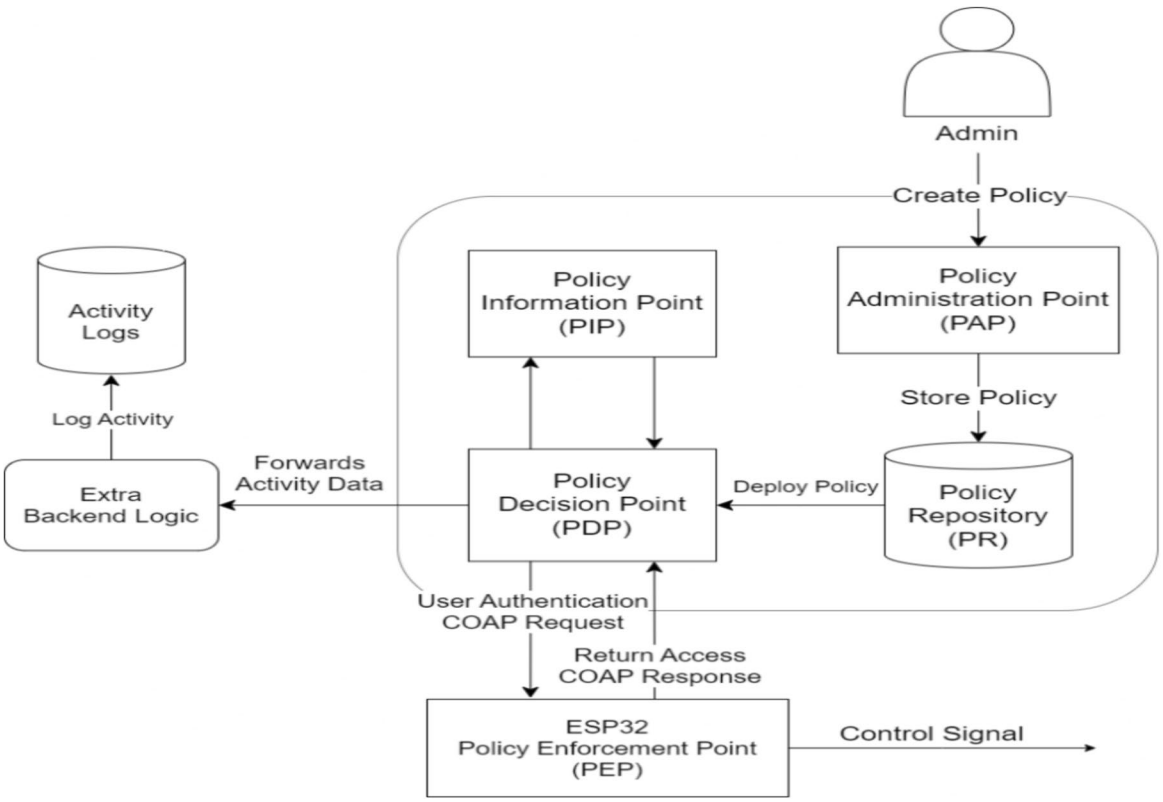


Fig. 2 Backend server ABAC software components block diagram

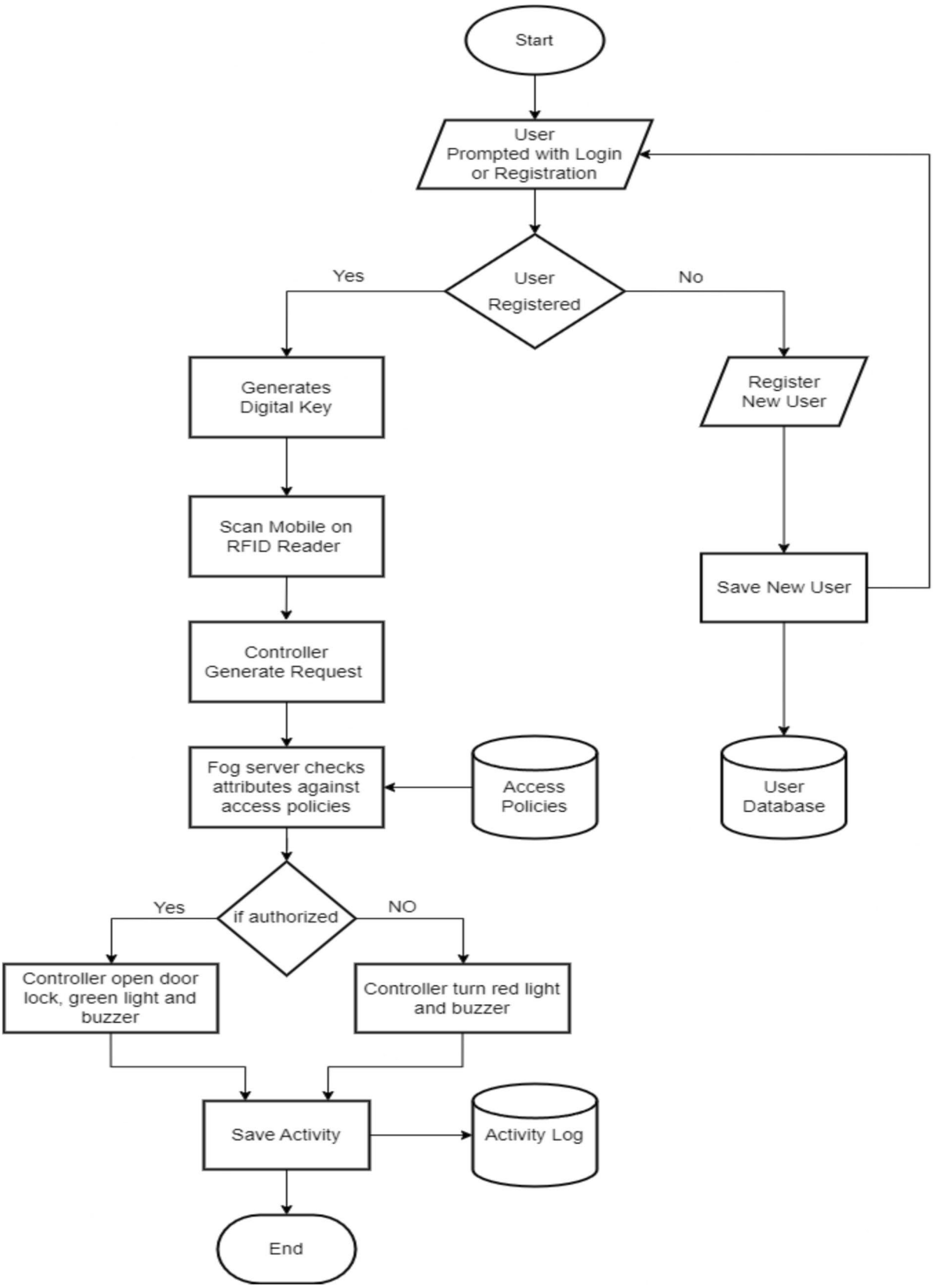


Fig. 3 ABACS flowchart

simply log in with their respective email and password credentials. Their records will be retrieved from the user database.
2. Upon login completion, the mobile application receives the user attributes from the server and initiates the secure generation and storage of the digital key for access control.
3. Then, users can present their mobile device to the access point readers to transfer their one-time-use digital key to the device through NFC communication.
4. Once the digital key is received, the device's controller processes the digital key and generates an access request that includes the digital key and access point attributes. The newly created digital key is sent to the backend server for thorough verification against the predefined access point policies.
5. The access request's attributes are extracted and checked against an access policy that is fetched from the policy store. After a decision is made, the backend server promptly communicates the decision back to the microcontroller, detailing the outcome of the validation process.
6. Based on the server's decision, the microcontroller executes access control measures:
    a. Authorized Access: The door mechanism is activated, accompanied by the illumination of a green LED and an audible buzzer signal.
    b. Unauthorized Access: Activation of a red LED and a distinct buzzer alert signify denial of access.

In both cases, the local server meticulously logs the access attempts for auditing purposes.

3.8.3 Sequence diagram

The sequence diagrams are presented to show the interaction between the various objects and components within the system in a chronological order. The components communicate through messages going back and forth along with arrows to depict the communication direction and the type of message being passed. This time around, the flow is shown in greater details to aid in better understanding the system functionalities. Figure 4 shows the interaction between the administrator and the local server to create and deploy access policies. First, the administrator is required to authenticate himself through login. Once authentication is done, he can design access policies, validate them, and deploy them on the server as needed.

Figure 5 shows the interaction between the user, mobile device, and local server when the user attempts to login or register onto the system. First, the user either chooses to login or register and enters his respective personal information on the mobile device. The mobile device establishes a TLS communication with the server and sends the user credentials. It then receives the authentication decision from the server. Depending on whether the decision is positive or negative, the mobile device receives different payloads. If authentication is successful, the mobile device receives a seed for nonce generation along with the user attributes which are stored after encryption inside the mobile device's memory. Finally, the user is granted access to the application. Otherwise, if the authentication decision is negative, nothing is sent to the mobile device and access to the application is revoked to the user.

Figure 6 shows the interaction during access control between the mobile device, the access point, and the local server. First, the mobile device generates a sequence of nonces based on the seed previously received. It then retrieves the user attributes from its storage, decrypts them and appends the generated nonce to them. The digital key is now ready and can be sent through NFC to the access point. The access point, constantly emitting a detection radio signal, detects the mobile devices and receives the digital key. Once received, the digital key is again modified to add the access point attributes retrieved from the microcontroller's non-volatile storage. Once the payload is ready, the payload is attached to a CoAP request, which is then sent through a DTLS channel. The server receives the access request, parses it, evaluates it against the access policy and responds back to the access point with its decision. Depending on the decision, the access point will either open the door or keep it closed.

3.8.4 User scenario

An employee or visitor with a pre-existing record in the company's database installs the ABACS mobile application and begins the registration process Fig. 7a. During registration, the employee or visitor submits identity details, which are securely transmitted to the local server via asymmetrically encrypted TLS over HTTP. In the Visitor scenario, the user must activate visitor mode during registration, bypassing the need to enter an employee ID. Upon successful registration, a digital certificate is stored on the user's mobile device. The ABACS backend verifies the credentials against the company's database, and upon successful verification, the user attributes are mirrored in the ABACS database and securely stored on their device. To request access, the user first opens the application and is prompted to provide his fingerprint as a first-factor authentication method. Then, he logs into the ABACS mobile application Fig. 7b using his identifying credentials (email and password) as a second-factor authentication method. The mobile device authenticates the ABACS server and establishes a TLS-encrypted

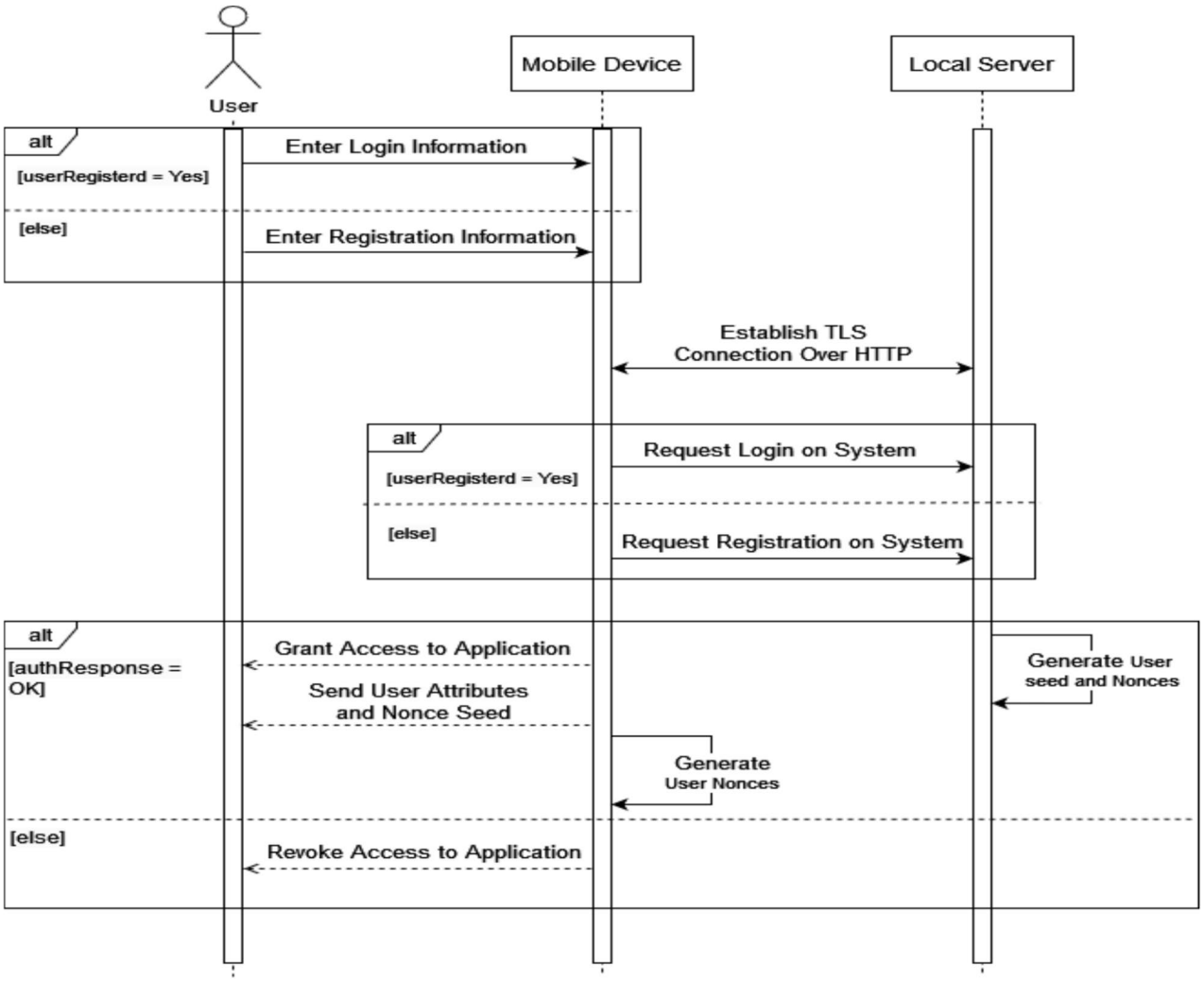


Fig. 4 Access policy development sequence diagram

session. The ABACS server generates a unique seed based on date and time, derives a sequence of 10 nonces, and sends the seed along with user attributes to the mobile device. The device then creates an active session by generating the same 10 nonces. It also encrypts the attributes using its TEE and stores them securely. At this point, the device is ready to initiate the access process Fig. 7c. When the user reaches the access point, the application retrieves the stored attributes and the next available nonce, generating a digital key. This digital key is transmitted via NFC to an access point. The access point appends its own attributes, stored in a JSON file, and securely transmits the data to the local server using CoAP over DTLS.

The local server evaluates multiple factors, including user attributes, access point attributes, environmental conditions, and predefined access policies. It verifies the nonce received from the mobile device against the previously generated sequence to ensure the correct seed was received during login, effectively serving as an OTP mechanism. If verification is successful, the system grants access and unlocks the door; otherwise, access is denied, ensuring strict security compliance. This structured approach ensures secure user access while maintaining data integrity and policy adherence.

4 ABACS implementation

The backend of the proposed ABACS relies on the microservices design architecture, where each microservice is independent from the other and communicates using API requests. This paradigm enforces the separation of concerns principle and drastically improves development, debugging and scalability aspects. Each microservice is responsible for implementing a set of related features that can be logically containerized within the microservice. All microservices

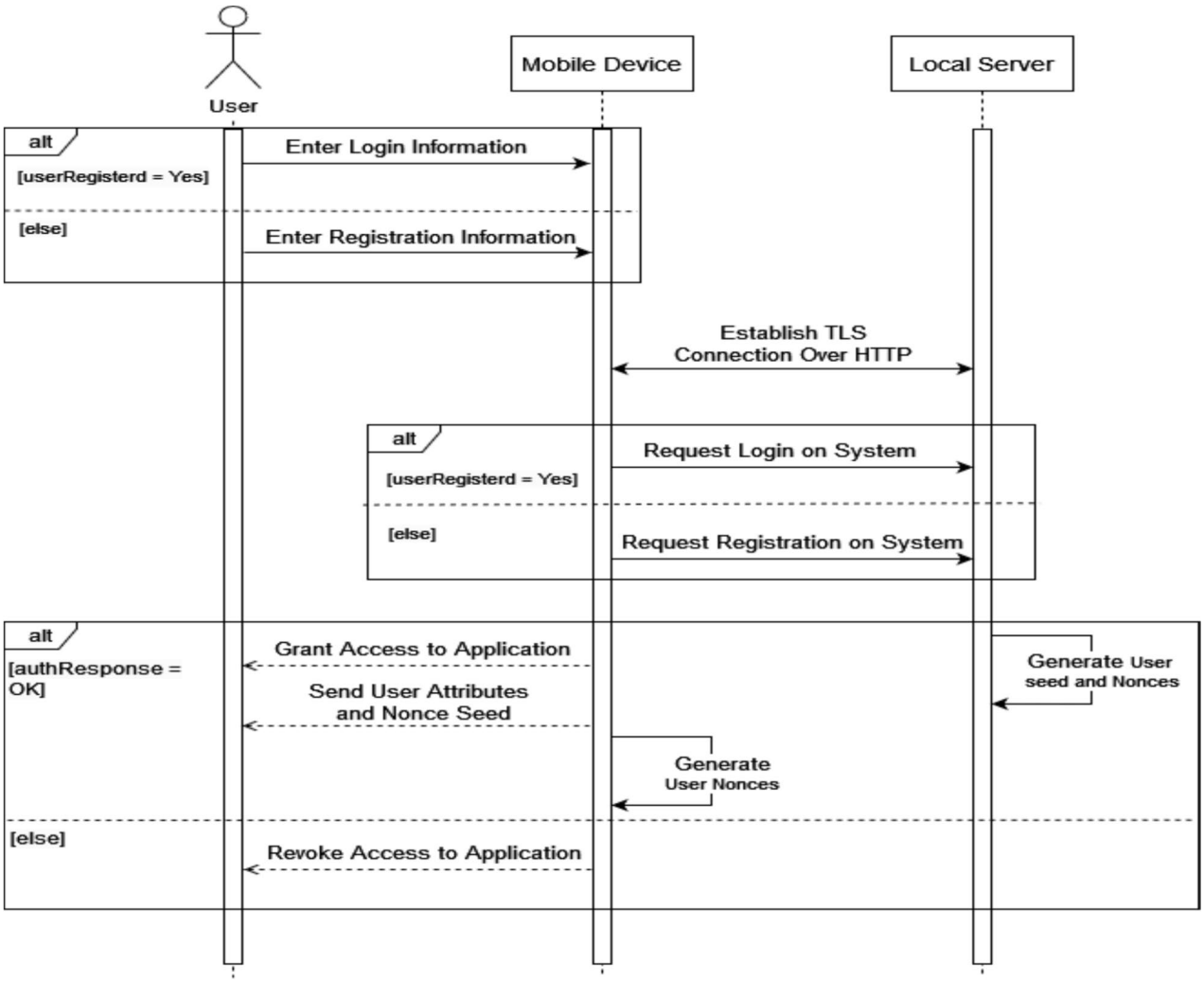


Fig. 5 Login and registration sequence diagram

are implemented using the RESTful design pattern, which offers simplicity and scalability, operating over the widely adopted HTTP protocol. Additionally, REST's compatibility with various devices and programming languages ensures seamless integration across different technologies [15]. The backend server will be deployed on a prototype server that must be capable enough to provide great servicing of requests and rapid response times, for that the Raspberry Pi 4 Model B was selected for the IoT backend server with its 64-bit CPU cores and up to 8 GB of RAM, ideal for running multiple internet services simultaneously. Combined with its community support, versatile connectivity, and GPIO pins, it's the best choice for scalable IoT solutions [16].

4.1 ABACS architecture decisions

The architectural decisions for the proposed ABACS are based on the adoption of the ABAC model over the RBAC for the backend server. The model was intentionally chosen for its ability to address limitations like role-centricity, lack of fine-grained control, and scalability issues. Leveraging ABAC, the proposed system incorporates dynamicity and detailed access policy definitions, enhancing management efficiency. TLS for HTTP and DTLS for CoAP are adopted to fortify internet communication, ensuring privacy, data security, and reliability. Additionally, the CoAP enforces good practices to enable communication between constrained-resource devices and addresses the specific constraints of IoT networks. The utilization of the TEE within the mobile device ensures secure data processing and protection, addressing critical security concerns. The system's software stack includes a versatile array of tools and frameworks for server, embedded system, and mobile application development, purposefully chosen to ensure robustness, scalability, and compatibility. Hardware component selections are meticulously made, aligning with performance,

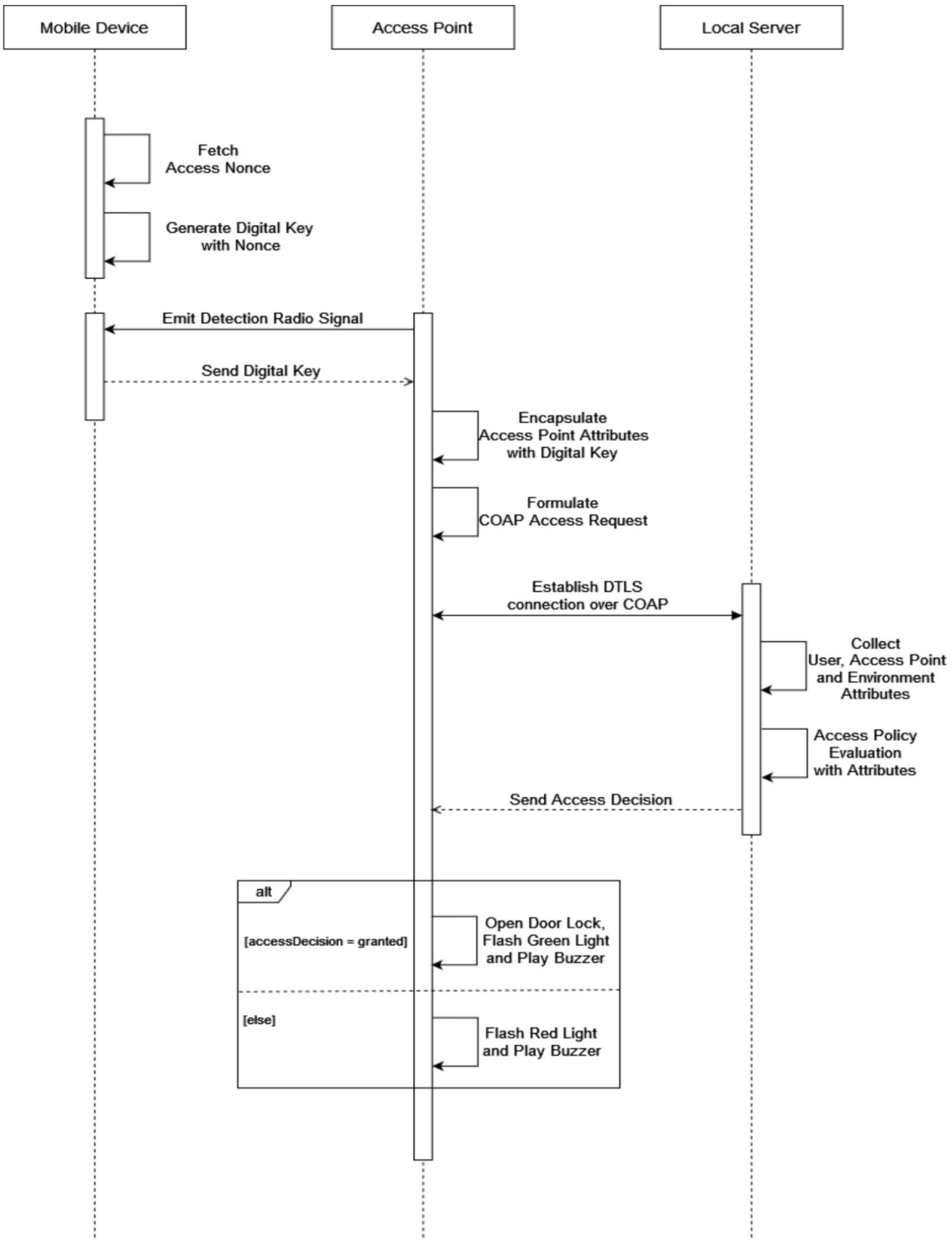
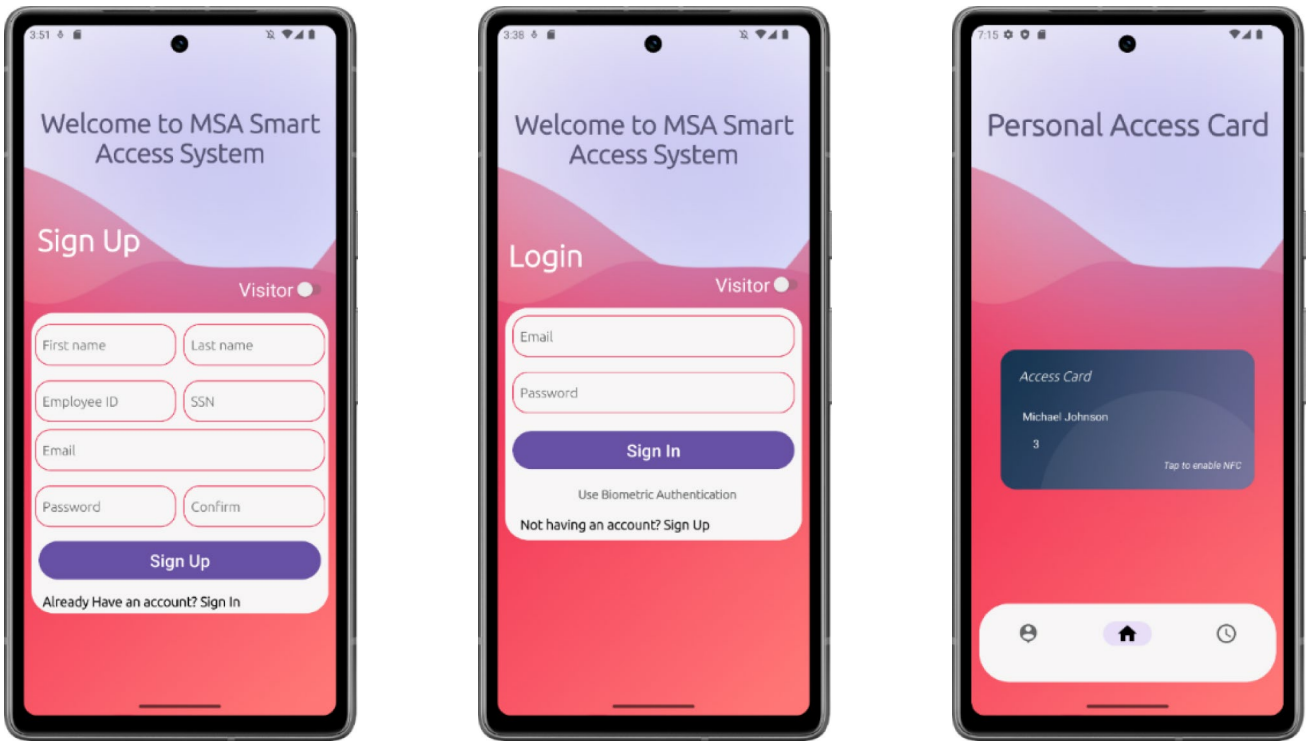


Fig. 6 Physical access control sequence diagram

connectivity, and security requirements. These architectural decisions collectively contribute to the robustness, scalability, and security of the proposed ABACS.

4.2 ABACS deployment strategy

The backend system must be deployed on a local server to manage and receive all the access requests. The first deployment strategy was to implement a service registration



(a) (b) (c)

Fig. 7 Mobile application **a** login page, **b** registration page and **c** home page

mechanism which provides multiple advantages, especially in terms of scalability. First, it provides automatic service detection at runtime, which eliminates the need for hardcoded endpoints and port numbers. This behavior is essential, especially in scaled environments where duplicate instances can be created at runtime. In addition, service registration also performs load balancing, which is a mechanism for even distribution of request traffic across services. Finally, service registration facilitates communication between all services as it constitutes a central hub for communication and service instances identification.

The second employed deployment strategy is the API gateway. It provides a unique entry point to the system, which prevents microservice instances from communicating with external clients in an uncontrolled manner. In addition, the gateway constitutes a unique secure entry point, which can authenticate and verify all incoming requests.

As part of the third deployment strategy, containerization was utilized to package each microservice and its dependencies into isolated execution environments known as containers. This approach ensures consistent behavior across various hardware and software environments, as each container includes all necessary libraries, runtime, and configurations. Containerization promotes lightweight virtualization and is particularly effective when applied within a microservices architecture, where each service operates independently. In ABACS, Docker is used to implement this strategy, enabling the backend to run as a set of modular, reproducible containers. These containers are orchestrated and monitored using Portainer, a user-friendly interface that integrates with Docker Compose and Docker Hub to manage container lifecycles, monitor performance, and automate deployment processes. This design offers several advantages for large-scale deployment, including horizontal scalability, where services can be replicated dynamically within and across fog nodes or cloud instances; fault isolation, which prevents the failure of one component from affecting the entire system; and portability, allowing deployment on a range of infrastructures. For instance, in the proposed prototype, the entire system is deployed and runs on the Raspberry Pi 4 Model B, which has been chosen for its large memory capacity of 8 GB of RAM and various connectivity options like Gigabit Ethernet and 802.11ac Wi-Fi. While this local deployment serves to validate system performance at the prototype level, the system was originally designed with a fog-cloud architecture in mind, for scalable, real-world implementations, where fog servers in each building

manage real-time access logic and a centralized cloud infrastructure handles remote coordination, analytics and policy distribution.

In the proposed architecture, the ABACS system is structured as a two-tier model composed of multiple fog nodes and a centralized public cloud. The first layer consists of local servers or also called fog nodes, deployed in individual buildings to host the core access control logic. These nodes handle latency-sensitive operations such as access validation and policy enforcement, ensuring real-time responsiveness, localized data privacy, and autonomous functionality even during temporary disconnections from the internet. The second layer comprises a centralized public cloud that serves as the global management hub, overseeing a distributed network of fog nodes. It enables remote monitoring, centralized policy updates, and analytics aggregation across buildings. A dedicated management software is envisioned to coordinate this layered infrastructure. It would support tasks such as supervising the health and performance of each node, synchronizing access control policies, and securely collecting operational metrics and system-wide logs. This division of responsibilities enhances system scalability, resilience, and visibility across multiple deployment sites.

To support this architecture effectively, each fog node should meet certain hardware requirements, including multicore processors (e.g., quad-core or higher), at least 8 GB of RAM, and sufficient local storage for logs and user data. These specifications ensure the ability to process concurrent access requests and manage communication with the backend and client devices. For the public cloud, the infrastructure should align with industry-grade standards, featuring virtual machines or containers with scalable CPU and RAM allocations, alongside high-speed network interfaces to manage incoming data from numerous fog nodes. A minimum dedicated bandwidth of 1 Gbps is recommended to facilitate seamless inter-building communication. Additionally, robust network security measures—including encryption protocols and firewall configurations—are essential to protect system integrity and prevent unauthorized access. Continuous monitoring and maintenance of both hardware and software components are crucial to uphold optimal performance and reliability. By adhering to these technical guidelines and leveraging the fog-cloud paradigm, ABACS ensures secure, scalable, and responsive deployment across modern smart building environments.

4.3 ABACS attributes and payload design

Each entity within the system has a set of attributes that is used during the access request evaluation to formulate a decision. The attributes were selectively chosen according to their relevance to the access request evaluation. For instance, user attributes include user ID, role, department, time schedule, clearance level and user status. Beside the ID, which is used for auditing purposes, all the attributes contribute to the access request decision-making. The proposed system includes multiple subsystems that exchange HTTP, NFC, and CoAP payloads. During user login and registration, the mobile device communicates with the backend server via HTTP, receiving a payload with user attributes. The mobile device then sends these attributes and an access nonce to the embedded access point system via NFC. The access point appends its own attributes and sends a CoAP request containing all the attributes. To optimize communication speed and latency, it is crucial to reduce the payload format and size. Consequently, a set of rules were applied on the attributes contained inside the various payloads to avoid sending unnecessarily large and redundant information. First, size constraints were applied on all attributes to prevent them from exceeding a limit of 20 characters, where one character has a size of one byte. Secondly, the JSON payload is structured in a key-pair fashion, where keys represent the attribute name, and the value represents the actual information contained within the attribute. The attributes keys were condensed to a maximum of two or three characters to minimize the overall size of the JSON payload and only include non redundant information.

4.4 Embedded system design and implementation

Embedded system for the Access point is designed using ESP32 microcontroller that is capable of handling internet connection with the backend server and capable of interfacing with all the other hardware components. Its built-in Wi-Fi capabilities, which allows for wireless communication with the backend server of the proposed IoT system. Its 34 General-Purpose Input–Output (GPIO) pins enable direct interfacing with many components without additional hardware. Its dual cores running at up to 240 MHz ensure smooth operation and efficient management of tasks. The embedded system was programmed inside the ESP-IDF framework with a feature called “Arduino as a component” which allows it to integrate and call Arduino-specific functions from the ESP-IDF framework.

4.5 Mobile application design and implementation

The Android application adheres to the Model-View-View-Model (MVVM) architecture pattern. Its essential benefits are a clean separation of concerns, scalability, reusability, maintainability, and testability. This pattern separates the application into three core layers, namely the UI, business logic, and data layers, which promotes clean architecture, making the codebase more modular and easier to understand [17]. The application is decomposed into multiple layers of development. First, the data layer allows the mobile

application to communicate through HTTP, the Retrofit client was used and allows for efficient communication with the remote backend server. In addition, the "BackEndApi" interface defines the HTTP endpoints to be requested from the backend server. It abstracts away the implementation details of network communication and provides a clear and standardized interface for interacting with remote resources. Furthermore, the "BackEndApi" interface acts as the principal gateway for making API requests. Secondly, the Plain Old Java Objects (POJO) layer is a directory within the application's directory structure, that serves as a repository for simple objects representing various data models crucial for communication with the backend. The collection of models ensures streamlined data exchange between the client and server. By centralizing these models within the "pojo" directory, the application promotes code organization, maintainability, extensibility, and adaptability to evolving backend requirements. Finally, the Presentation layer where the application's user interface (UI) components exist, are meticulously structured within the "UI" directory. Activities such as main, signup, sign in, home, and profile, all reside in this directory. Each activity is designed with a specific user interaction flow in mind.

5 ABACS testing results and evaluation

5.1 Security testing and robustness

The ABACS system was designed with a multi-layered security architecture to address threats across communication, application, physical, and NFC interfaces. This section presents a comprehensive threat analysis based on current attack taxonomies in physical access control and NFC-enabled systems, incorporating categories of network-level, application-level, device-level, and NFC-specific attacks. Each subsection evaluates the corresponding countermeasures implemented in ABACS, identifies limitations, and outlines future mitigation plans when applicable.

5.1.1 Network-level attacks

Network-level attacks target the communication between system components such as the mobile application, backend server, and access controller. These threats are particularly relevant in distributed systems like ABACS, where sensitive data (e.g., digital keys, session tokens) is exchanged over HTTP or CoAP protocols. Attackers may attempt to intercept, alter, or replay messages as they traverse the network. To counteract these threats, ABACS integrates modern transport security protocols and enforces strict session handling policies to ensure confidentiality, integrity, and authenticity during data transmission.

5.1.1.1 Man-in-the-middle attacks Man-in-the-middle (MITM) attacks aim to intercept or alter communications between two trusted entities, often to impersonate one party or extract sensitive data. In the context of ABACS, such attacks could target the communication channel between the mobile application and the backend server, or between the access controller and the local service. To mitigate this threat, ABACS secures all HTTP-based communications using TLS 1.3 and applies DTLS for CoAP-based constrained environments. These protocols provide encrypted transmission, mutual authentication, and message integrity. As described by Razumov et al. [11], the handshake and record protocols in TLS are fundamental for resisting MITM attacks by establishing session-specific keys and verifying digital certificates. During implementation, a self-signed certificate is used to authenticate endpoints within a local, controlled environment. While this setup does not offer the full trust chain of a CA-issued certificate, it remains effective for closed prototype deployments where the certificate is securely distributed and verified. Razumov et al. [11] emphasize that even in TLS-secure sessions, user mismanagement of certificates, such as accepting invalid or spoofed certificates, can still open the door to MITM attacks. Therefore, migration to CA-issued certificates in production environments is recommended to improve trust and resilience. Within its scope, the ABACS deployment leverages the TLS stack to preserve the confidentiality and authenticity of all transmitted access credentials. In short, ABACS works as follows: a user opens the ABACS app and requests access to a restricted door. As the request is transmitted to the backend over TLS, a malicious actor on the same network attempts to intercept the message. However, because the session is encrypted and authenticated with a certificate, the intercepted data is unreadable and cannot be altered or replayed by the attacker.

When comparing ABACS against existing systems, it is noticed that Arnosti et al. [4] partially addresses MITM attacks using a microSD-based Secure Element that performs cryptographic operations independently but still relies on the mobile OS to initiate TLS sessions, which introduces risk. Noprianto et al. [7] does not mention TLS or SSL, relying solely on application-layer authentication through JWT and SHA-256, which cannot prevent message interception. Reza et al. [6] and Petrakis et al. [5] omit any discussion of encrypted transport protocols, leaving their systems exposed to interception and impersonation threats.

5.1.1.2 Internet replay attacks Replay attacks over the internet involve intercepting and resending previously captured HTTP or CoAP requests to trick the backend server into executing unauthorized actions. In ABACS, such attacks are mitigated using TLS 1.3 for HTTP-based communications and DTLS for CoAP communications in constrained

environments. As mentioned by Razumov et al. [11], these protocols not only encrypt data in transit but also provide session-specific cryptographic keys, message sequencing, and anti-replay protection built into the protocol itself. As a result, even if an attacker manages to capture a legitimate access request, the replayed message would be detected and discarded by the protocol layer before reaching the backend. This ensures message confidentiality, integrity, and freshness across all ABACS internet-facing communications. For example, an attacker captures a legitimate HTTP access request using a packet sniffer. Later, they attempt to replay the same message to gain access. However, the TLS session keys, and message sequence used in the original session are no longer valid, causing the backend to detect and reject the replayed message instantly.

Evaluating ABACS alongside other systems shows that Arnosti et al. [4] does not detail any freshness verification or nonce-based mechanisms, and while Noprianto et al. [7] employs JWT tokens, it does not include any replay prevention strategy such as timestamps or one-time-use constraints. As a result, replayed messages could be processed as legitimate requests. Reza et al. [6] appears to offer some level of replay protection through biometric verification and IMEI checks, although this is not cryptographically enforced. Petrakis et al. [5] lacks any form of session validation or nonce-based replay prevention.

5.1.2 Application-level attacks

Application-level attacks exploit flaws on the client side of the system, namely the mobile device in ABACS. Attackers seek to obtain unauthorized access to the mobile application to be granted access to the restricted area. To prevent this, ABACS's custom mobile software implements a multifactor authentication to verify the user's identity, prior to any interaction with the access point and decision enforcement.

5.1.2.1 Unauthorized mobile application access If an adversary gains physical access to a user's smartphone, they may attempt to open the ABACS app and generate digital keys to gain unauthorized entry. ABACS enforces a robust first line of defence by requiring biometric authentication, namely a fingerprint scan, before granting access to the app interface. This ensures that only the legitimate device owner can interact with the application. In addition to this local authentication, ABACS requires the user to log in to the application to initiate a session with the backend, which serves as a second layer of authentication by verifying the user's identity before allowing access to key generation features. This multifactor authentication approach ensures that the app remains secure even in scenarios where the device is shared, or a session remains active. Digital keys are never stored persistently on the device and must be freshly generated following successful backend authentication. The backend also validates session tokens and cryptographic signatures to ensure that key requests are legitimate and user-specific, reinforcing the principle that physical access to the device alone is insufficient to compromise the system. Commonly, a thief unlocks a stolen phone using a known PIN but is unable to access the ABACS app, which is locked behind a fingerprint scan. Even if they bypass this layer, they must log in with valid credentials to generate a key. Without access to both factors, the app and the system remain secure.

When placed side by side with prior systems, ABACS highlights that Reza et al. [6] implements fingerprint authentication, offering comparable protection, while Arnosti et al. [4], Petrakis et al. [5], and Noprianto et al. [7] provide no indication of in-app authentication or PIN mechanisms, leaving their mobile interfaces open to misuse if the device is compromised.

5.1.3 Physical and device-level attacks

Physical and device-level attacks involve tampering with the embedded hardware, exploiting insecure storage on user devices, or extracting cryptographic material through side channels. These attacks are especially relevant in IoT-based PACS systems like ABACS, where edge devices play a critical role in mediating access requests. ABACS reduces these risks by minimizing the authority of local controllers, securing cryptographic operations within hardware-protected environments, and ensuring that sensitive data is encrypted and bound to authenticated user sessions.

5.1.3.1 Physical tampering with embedded controllers Attackers may attempt to physically tamper with the ESP32-based embedded system that handles NFC communication and controls the door lock. ABACS addresses this risk by integrating a magnetic reed switch within the enclosure to detect unauthorized opening or displacement. When triggered, the system logs the event and disables access temporarily. Additionally, access decisions are made exclusively on the backend; the embedded controller simply acts as a relay for NFC-based requests. Thus, tampering with the local device yields no control over access logic or policy evaluation. A malicious actor opens the controller's casing to modify the hardware. The reed switch immediately detects the breach and logs it to the backend, which disables access to the door until the issue is reviewed by an administrator.

In contrast to the proposed system, none of the other systems include any mention of physical tamper detection or protective hardware mechanisms, leaving their embedded components exposed to unauthorized manipulation or reverse engineering.

5.1.3.2 Side-channel credential cloning In cases where the mobile OS is compromised, an attacker may attempt to reverse-engineer or extract stored credentials to impersonate the legitimate user. ABACS mitigates this risk through a layered data protection approach. Cryptographic keys used for encryption are generated and securely stored within the device's TEE, a secure hardware-backed component that ensures these keys remain inaccessible to unauthorized processes, even with root access. However, the actual access credentials, such as attribute values and nonce tokens, are stored outside the TEE, within the operating system's file storage. To protect this data, ABACS encrypts these files using the TEE-managed keys, ensuring that their contents remain unreadable without authorized decryption operations. As a result, even if an attacker gains access to the file system, the encrypted data remains unusable. Furthermore, ABACS enforces that each access attempt must undergo backend authentication and session validation, and that digital keys are dynamically generated and expire after use. This combination of hardware-backed key management, encrypted storage, and session-bound credentials makes side-channel cloning and credential extraction infeasible within the ABACS system. For example, a user's phone is infected with malware that attempts to extract access keys from the file system. However, the keys are encrypted using TEE-managed secrets. The attacker is unable to decrypt and use any extracted data.

Reviewing ABACS in the context of previous work shows that, for example, Arnosti et al. [4] offers comparable protection via the use of a tamper-resistant microSD-SE, which securely stores credentials and performs isolated cryptographic functions. Reza et al. [6], Petrakis et al. [5], and Noprianto et al. [7] do not include any mention of secure credential storage or hardware-backed isolation, making their systems susceptible to cloning if the mobile device is compromised.

5.1.4 NFC-specific attacks

NFC-specific attacks arise from the nature of short-range wireless communication used between mobile devices and physical access readers. In access control systems like ABACS, these interactions are central to how users present their credentials for authentication. ABACS incorporates several layers of protection to safeguard this interface, including backend validation and contextual checks.

5.1.4.1 NFC replay, skimming and cloning attacks In NFC-based access control systems, attackers may attempt to gain unauthorized entry by capturing and replaying a legitimate NFC message or by cloning a credential for reuse. These threats exploit the predictable and static nature of conventional RFID transmissions. ABACS effectively mitigates such risks by generating ephemeral digital keys that are valid for a single session only. Each key includes a cryptographic nonce and is tied to the user's authenticated session. Upon first use, the key is invalidated and cannot be replayed or reused. Moreover, ABACS never transmits fixed identifiers or permanent attributes, eliminating opportunities for skimming or duplication. This layered approach ensures that even if an attacker captures an NFC payload during transmission, the credential becomes contextually invalid immediately after use and is rendered useless for future attempts. This aligns with best practices for dynamic credential lifecycle management and conforms to recommendations from the NFC security literature [18].

For instance, as a user presents their phone to the NFC door reader, a hidden skimming device captures the transmitted key. Later, the attacker re-emits the same data using a rogue NFC device. However, the backend system detects that the session-specific key has already been used and instantly rejects the request, effectively neutralizing the attack.

A comparison between ABACS and existing systems reveals that Arnosti et al. [4] acknowledges the vulnerability of its HCE-based solution to NFC replay attacks, while its microSD-SE variant likely offers improved resistance due to hardware-based key storage and cryptographic signing. Reza et al. [6] relies on biometric and IMEI pairing for access control, which may incidentally limit abuse but does not explicitly prevent NFC-level cloning or replay. The systems by Petrakis et al. [5] and Noprianto et al. [7] transmit fixed credentials or identifiers, such as BLE beacon IDs or static RFID keys, without nonce-based validation or dynamic key rotation, leaving them exposed to skimming, duplication, and replay-based entry.

5.1.5 Remaining security considerations

While ABACS effectively mitigates many security threats, some residual risks remain. Eavesdropping on the NFC channel is theoretically possible if an attacker is in very close physical proximity. Although ABACS transmits only ephemeral, session-bound keys that expire after use, the absence of NFC-layer encryption still leaves a minimal exposure window. As noted by Onumadu & Abroshan [18], applying end-to-end encryption using standards such as AES or ECC would ensure that intercepted data remains unreadable, and should be considered in future iterations of the system.

Relay attacks, where NFC messages are forwarded in real-time to impersonate a legitimate user, are particularly dangerous due to their stealth and physical simplicity. ABACS mitigates this through nonce-based key generation, backend validation, and requiring active user app sessions. However, since ABACS currently lacks NFC-level proximity verification or cryptographic methods to tie access attempts directly to the user's physical context, advanced relay attacks

using external hardware may still be possible. To address this, future improvements may involve implementing biometric confirmation before key generation, and mutual authentication protocols, as recommended by Onumadu and Abroshan [18]. Lastly, data tampering remains a concern due to the lack of cryptographic message integrity checks on the NFC layer. Although ABACS rejects malformed or altered payloads through backend validation, attackers may still attempt to manipulate the structure or timing of requests. To enhance resistance to such manipulation, future updates will integrate lightweight cryptographic protections such as message authentication codes (MACs) or digital signatures. These measures are strongly recommended by Onumadu and Abroshan [18] to ensure payload integrity and protect against unauthorized modification.

5.2 Scalability testing and responsiveness

The proposed ABACS physical access control system can successfully process requests and grant, or revoke access based on users, access point and environment attributes. However, the system must be tested to verify that it can withstand a heavy traffic of requests and still respond in short periods of time. Consequently, as Petrakis et al. [5] mentioned in their research paper and proposed system (iPACS), the server's access control endpoint needs to be benchmarked to verify its behavior in high-traffic environments. The iPACS paper was selected as a primary comparison point due to its comprehensive and well-documented benchmarking procedure, which sets a solid foundation for performance evaluation while other recent similar benchmarks lack clear benchmarking data. As such, our benchmarking approach closely follows the methodology outlined in their work. The iPACS system utilized the traditional HTTP protocol to send access requests from users' mobile devices to the fog server. However, the proposed ABACS system relies on the CoAP protocol established between the access point unit and the local server. Consequently, the benchmarking procedure utilized different benchmarking tools and procedures. The Californium CoAP-Extplugtest software program was used to conduct benchmarking like the way Petrakis et al. [5] obtained their results. Along with the POST request and URI instructions, a payload file containing valid user and access point attributes was attached with the request. In addition, a file containing the PSK information was attached to authenticate the benchmark clients to the server. Key factors influencing the benchmarking outcomes included the backend server hardware, specifically, a Raspberry Pi 4 Model B equipped with a 1.5 GHz quad-core Cortex-A72 processor and 8 GB of LPDDR4-3200 SDRAM, and the benchmarking client, a desktop PC powered by an AMD Ryzen 7 3700X CPU (8 cores, 16 threads, 3.6 GHz base clock, 4.4 GHz boost clock) paired with 32 GB of Crucial Ballistix DDR4-3200 RAM. Network conditions were also critical; tests were conducted over a wireless connection, with upload bandwidth constraints applied using the NetLimiter software program to simulate various network scenarios.

Benchmarking was conducted in two phases. The first phase included sending 200 and 2000 requests with concurrency levels 1, 40 and 120, without restraining the upload data rate, to observe how the backend system handles rapidly incoming requests. The second phase included sending those same requests but with an upload data rate limit, to find out the minimum bandwidth required from the network installation to allow a proper communication between the access point and the local server. Finally, along the maximal servicing time per request percentile, the average CPU and memory consumption in the Raspberry Pi were also monitored using the Linux top command, which provides exhaustive details about system processes and resource utilization. Finally, data rates were defined using the "NetLimiter" software application. The results of the first phase of benchmarking are presented in Tables 1, 2, 3 and 4. The obtained average times per percentile are included, along the observed data rate and the CPU consumption. It is noticed

Table 1 200 CoAPS requests (concurrency = 1)

Percentage of served requests	95%	99%	99.9%	100%	Avg time (ms)	Avg CPU
Time taken at 3.83 KB/s (ms)	34	44	45	45	22.34	19.7%

Table 2 2000 CoAPS requests (concurrency = 1)

Percentage of served requests	95%	99%	99.9%	100%	Avg time (ms)	Avg CPU
Time taken at 21.36 KB/s (ms)	49	59	74	83	29.30	18.51%

Table 3 2000 CoAPS requests (concurrency = 40)

Percentage of served requests	95%	99%	99.9%	100%	Avg time (ms)	Avg CPU
Time taken at 62.16 KB/s (ms)	744	794	824	837	434.41	53.28%

that the system performs very well with different numbers of requests and at different concurrency rates. It achieves the targeted latency included in the non-functional requirements. Sequential requests had to be served under 1 s of average servicing time, which has been achieved, as shown in Tables 1 and 2, where the average time hovered between 20 and 30 ms only. In addition, when it comes to concurrent requests, the target of servicing all requests under an average time of 3 s has also been achieved. Tables 3 and 4 show that with high levels of concurrency, 40 and 120, the system still achieves a low average servicing time of 434.41 and 1367.22 ms respectively.

First, it is noticed that as the concurrency level rises, the data rate and CPU consumption also rise to meet the demand of sending and processing a high number of incoming requests. It is also worth noting that the CPU consumption does not increase as the number of requests increases in sequential scenarios, most likely because requests are always processed one at a time and increasing their number will not result in a change in the behavior of the system. The system needs to be benchmarked at lower data rates, to evaluate its performance under constrained networks and evaluate the minimum required data rate that still achieves the objective target average times of 1 and 3 s in sequential and concurrent scenarios respectively. To limit the network speed, the NetLimiter software program was used. In every scenario, two graphs were used to represent both the average servicing time and the corresponding CPU consumption. Results are presented in the form of linear graphs in Figs. 8 and 9.

In Fig. 8a, when benchmarking the backend system with 200 requests and concurrency 1, it is noticed that the system achieves the target average time of 1 s between data rates of

Table 4 2000 CoAPS requests (concurrency = 120)

Percentage of served requests	95%	99%	99.9%	100%	Avg time (ms)	Avg CPU
Time taken at 56.63 KB/s (ms)	2494	2804	2859	2871	1367.22	62.22%

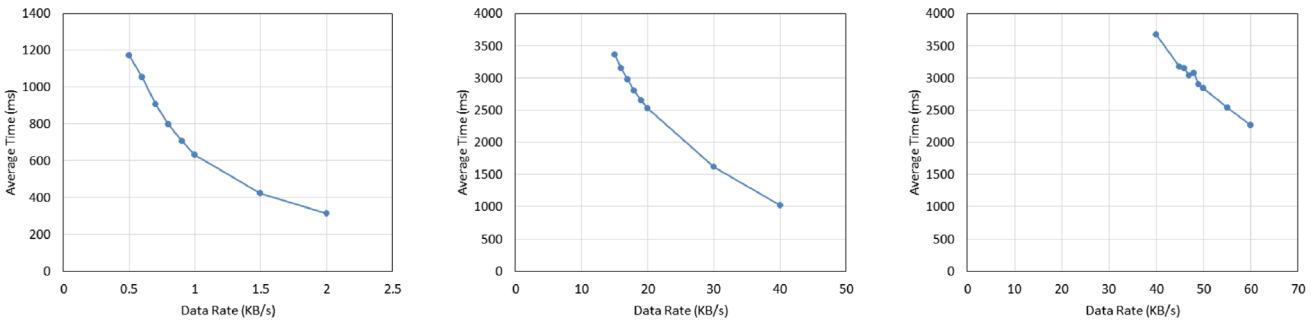


Fig. 8 Data rate impact on average servicing time for **a** 200 requests [concurrency = 1], **b** 2000 requests [concurrency = 40] and **c** 2000 requests [concurrency = 120]

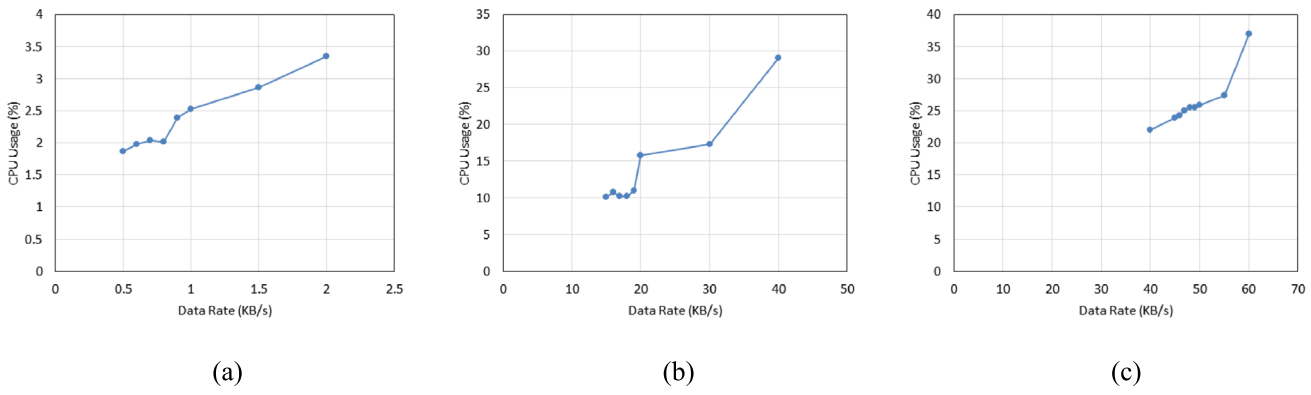


Fig. 9 Data rate impact on CPU usage for **a** 200 requests [concurrency = 1], **b** 2000 requests [concurrency = 40] and **c** 2000 requests [concurrency = 120]

0.6 and 0.7 Kbytes per second. With the slight variations and inconsistencies that happen in network quality and latency, it is safe to assume that the proposed system's breakeven data rate is around 0.7 Kbytes per second in sequential scenarios. Also, Fig. 9a shows that low data rates result in minimal CPU usage, which increases only linearly as the network rate grows. As a result, this indicates that sequential requests are lightweight to process and place minimal demands on both network bandwidth and processing power.

In Fig. 8b, when running 2000 requests with concurrency 40, there is a clear increase in the minimum data rate required to achieve the target time of 3 s maximum. In fact, it seems that around 17 Kbytes per second are needed to provide an average servicing time of 3 s. This is explained by the fact that requests are now sent concurrently, which overloads the network infrastructure and necessitates higher speeds to successfully transfer all the requests from the client to the server. In addition, Fig. 9b shows that the backend system also needs to process all the incoming requests in a concurrent manner, which consequently increases the CPU consumption as the data rate increases as well.

Finally, in Fig. 8c, when running 2000 requests with concurrency 120, the minimum data rate threshold increases even more and seems to hover around 49 Kbytes per second, which is again explained by the sheer number of requests that saturate the network and demand a higher rate to be successfully transferred without errors. The CPU consumption also increases along the data rate proportionally as shown in Fig. 9c.

It is important to compare ABACS performance to an existing system, like iPACS, to evaluate the impact of the use of CoAP and the microservice backend implementation on similar access control systems. Figure 10 represents the longest performing request of iPACS against ABACS. The proposed system was also benchmarked with the same data rates observed in the iPACS benchmarking to closely compare the obtained results. The data rates used were relatively low. The data rates are 0.61 KB/s for both sequential scenarios represented in Fig. 10a, 7.04 KB/s with concurrency 40 and 7.61 KB/s with concurrency 120 in Fig. 10b. Consequently, it is noticed that the proposed system's comparative performance varies a lot and heavily depends on the network data rate. When comparing both systems at similar low bandwidths, the proposed system performs slightly better than iPACS in sequential scenarios. However, when processing concurrent requests at the same low data rates, iPACS outperforms the proposed ABACS system. This is likely due to the difference in payload format and size between the two systems. Reliance on the ABAC model means that all access point and user attributes must be sent within the request payload. As mentioned in previous sections, a significant advantage of using the ABAC model over the RBAC model is its flexibility and granularity in access control. ABAC can consider a wider array of attributes, such as user properties, resource characteristics, and environmental conditions, which allows for more fine-grained and context-aware access decisions. It also enhances security by ensuring that access decisions are made based on comprehensive information rather than just predefined roles.

On one hand, the ABAC model certainly provides more versatility but on the other hand, it also noticeably increases the payload size, even though it was largely optimized and reduced. A larger payload size needs a higher bandwidth to be promptly sent without connection timeouts. On the other hand, the iPACS relies on the RBAC model, which bases its decision-making on the code area and the user role only; therefore, reducing the amount of information needed inside the payload. Unfortunately, details about the payload format

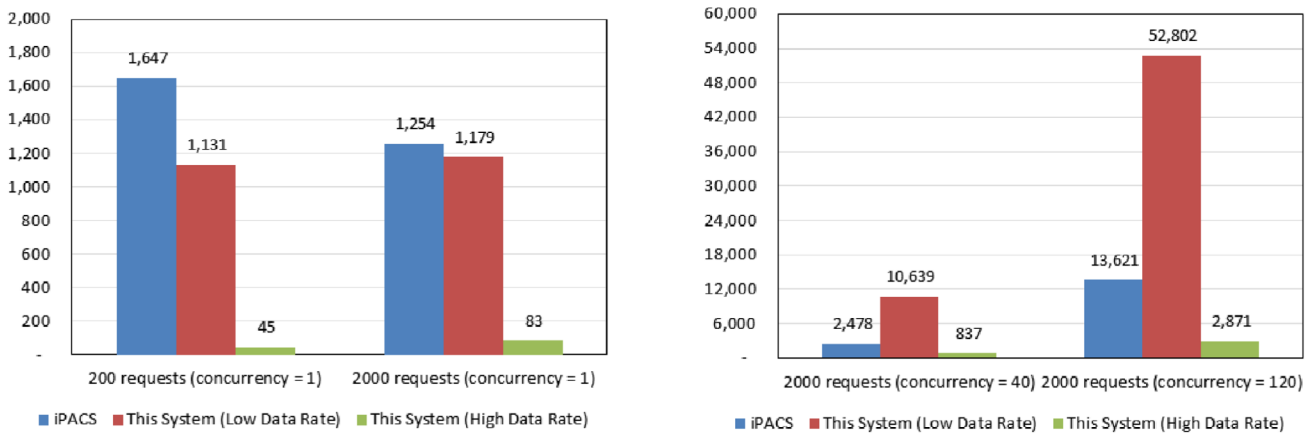


Fig. 10 System comparison of longest performing request for **a** [concurrency = 1], **b** [concurrencies 40 and 120]

and size were not exposed in Petrakis et al.'s work [5]. Nonetheless, when removing data rate limits, it is noticed that the proposed system becomes much more efficient and outperforms iPACS. This confirms that data transmission rates and payload sizes are the main factor behind the system's performance against iPACS's. Once communication rates are suitably adapted to ABACS's payload requirements, it is noticed that the proposed system's backend efficiently processes requests in both sequential and concurrent scenarios. With sequential requests, the proposed system's longest-performing request outperforms iPACS's by at least tenfold. With concurrent requests, ABACS shows a performance at least twice better than iPACS's.

The main communication protocol in the proposed system between the embedded client and the server is conducted via CoAP instead of HTTP. The CoAP protocol reduces the header size and optimizes the bandwidth consumption, which makes it more efficient than the HTTP protocol. To support these claims, the benchmarking was reattempted on the proposed system with both HTTP and CoAP protocols. The CoAP request is sent to the CoAP server, which in turn communicates with the access control microservice, while the HTTP request is sent to the API gateway, which communicates with the access control microservice too. Both protocols use a secure channel, DTLS and TLS respectively. The results are shown in Table 5.

It is noticed that the CoAP communication protocol performs better than the HTTP protocol given the same data rate and communicated payloads. First, the difference between the performance of both protocols in sequential requests is very small, because there is effectively no stress being put on either the server nor the network to communicate and process the request; but the CoAP protocol still sends less data and hence, performs better by a small margin. Secondly, when evaluating concurrent requests, the CoAP protocol surpasses the performance of the HTTP protocol by a large margin. Due to the heavy load put on the network, it is essential that payloads be minimized so that they can be transferred more quickly, and the CoAP protocol achieves this, as demonstrated by the obtained results.

5.3 User-friendliness evaluation

ABACS prioritizes user-friendliness without compromising security by offering a seamless and intuitive workflow from registration to access. New users, whether employees or visitors, can register directly through the mobile application without requiring physical interaction with the system administrator or manual provisioning. Employees benefit from automatic identity verification against a pre-existing database, while visitors can activate visitor mode, simplifying temporary access enrollment. The application combines biometric authentication and login credentials in a logical sequence, offering a dual factor experience that feels natural and frictionless. Once authenticated, users no longer need to interact with the backend directly, digital keys are generated in the background using stored credentials and time-based nonces and transmitted via NFC with a single tap at the access point. The entire access process is designed to require minimal user input after login, providing a smooth experience while maintaining tight security controls.

When compared to existing systems, ABACS demonstrates superior user-friendliness. Arnosti et al. [4] relies on external microSD-based secure elements, which require hardware compatibility and additional user handling during setup,an approach that limits adoption on modern smartphones. Reza et al. [6] depends on device-specific IMEI validation, which requires the user to manually input the IMEI number into a physical keypad attached to the access controller during registration. This step is cumbersome, especially when users are unfamiliar with their IMEI or when multiple users attempt first-time registration simultaneously. Petrakis et al. [5] incorporates a passive BLE-based access mechanism and notifies users via push alerts if unauthorized entry is attempted. However, the system lacks real-time enforcement,it does not prevent entry at the physical level before the violation occurs, which limits its effectiveness. Noprianto et al. [7] uses RFID cards for user identification, but physical cards present well-known drawbacks: they can be forgotten, lost, stolen, or damaged. In contrast, mobile phones on which ABACS operates are rarely forgotten and benefit from native biometric and session-level authentication. ABACS integrates familiar smartphone interactions, real-time cryptographic safeguards, and backend intelligence into a frictionless and context-aware user journey.

Table 5 Performance comparison between CoAP and HTTP (average request servicing time)

	HTTP	CoAP
200 requests with concurrency 1 (in ms)	61.43 at 10.77 KB/s	53.99 at 10.77 KB/s
2000 requests with concurrency 1 (in ms)	54.26 at 12.59 KB/s	49.69 at 12.59 KB/s
2000 requests with concurrency 40 (in ms)	1257.82 at 47.14 KB/s	768.10 at 47.14 KB/s
2000 requests with concurrency 120 (in ms)	5521.77 at 30.55 KB/s	4947.14 at 30.55 KB/s

6 Conclusion

This paper introduced ABACS, a containerized Attribute-Based Access Control System that integrates the ABAC model with a modular microservice architecture, secure communication protocols, and a mobile-based authentication mechanism. The system addresses key limitations in existing PACS, including the lack of scalability, fine-grained policy enforcement, and robust security measures, particularly in resource-constrained and IoT-based environments. A structured security evaluation demonstrated that ABACS is resilient against several critical threat vectors, including man-in-the-middle attacks, internet replay, unauthorized mobile application access, NFC cloning, skimming, and physical tampering. These protections are achieved through the integration of TLS/DTLS communication protocols, per-session nonce generation, biometric and credential-based authentication, and secure key management within TEE. In contrast to earlier systems, which either partially address or overlook such vulnerabilities, ABACS adopts a layered security approach that strengthens both network and device-level protection. The system's architecture was implemented using a containerized microservice design, supporting modularity, fault isolation, and scalability. While the current deployment was conducted on a local server for prototype validation, a fog-cloud architecture was proposed as a future deployment model. This layered architecture would allow latency-sensitive operations to be handled locally at fog nodes, while delegating centralized management and monitoring functions to a cloud-based backend. Although not implemented in the present work, this deployment strategy is conceptually aligned with large-scale, distributed environments such as smart buildings or city-wide access infrastructures. Performance evaluations confirmed that ABACS remains responsive under both sequential and concurrent request loads. Benchmarking demonstrated that, under sufficient network bandwidth, the system outperforms comparable solutions such as iPACS in terms of request handling time and resource utilization. The adoption of the CoAP further reduced communication overhead, making the system suitable for deployment in constrained IoT contexts. In summary, ABACS constitutes a comprehensive access control framework that combines fine-grained policy enforcement with secure communication and scalable system architecture. Future work will focus on enhancing the resilience of the NFC communication layer through cryptographic message integrity checks and proximity-aware authentication methods, as well as the implementation and evaluation of the proposed fog-cloud deployment model in real-world environments.

Authors contributions Samer I. Mohamed took the ownership of the proposed system conceptualization, methodology, writing original draft preparation, supervision and responding to reviewers feedback. Manal Mostafa for the data collection, formal analysis, writing review and editing. Ahmed Salah for the software components design, visualization, investigation, validation. Jalal Assaly for the hardware components design and implementation, integration testing for the proposed system.

Funding We don't have any funding support as part of this manuscript development.

Availability of data and material The dataset used is part of the cited benchmark and is referenced in the manuscript for any future needs.

Declarations

Conflict of interest We don't have any kind of conflicts of interest as part of this manuscript with any party.

References

1. Huang P-C, Chang C-C, Li Y-H, Liu Y (2017) Efficient access control system based on aesthetic QR code. Springer, London, p 11. https://doi.org/10.1007/s00779-017-1089-y
2. Moore J (2022) The 2022 state of physical access control report. IFSEC Global
3. Aftab MU, Qin Z, Zakria, Ali S, Pirah, Khan J (2018) The evaluation and comparative analysis of role-based access control and attribute-based access control model.: In Proceedings of the IEEE international conference. IEEE, pp 35–39. https://doi.org/10.1109/ICCWAMTIP.2018.8632578
4. Arnosti C, Gruntz D, Hauri M (2015) Secure physical access with NFC-enabled smartphones. In: Proceedings of the 13th international conference on advances in mobile computing and multimedia. ACM, New York, NY. https://doi.org/10.1145/2837126.2837138
5. Petrakis EG, Antonopoulos F, Sotiriadis S, Bessis N (2020) iPACS: A physical access control system as a service and mobile application. J Ambient Intell Human Comput 11:929–943. https://doi.org/10.1007/s12652-019-01205-5
6. Reza R, Alam A, Islam ME (2019) IoT and Wi-Fi based door access control system using mobile application. In: 2019 IEEE international conference on robotics, automation, artificial intelligence and internet-of-things (RAAICON. IEEE, pp 9–12. https://doi.org/10.1109/RAAICON48939.2019.09
7. Noprianto N, Wakhidah R, Ariyanto R, Syaifudin YW (2023) Door access system design using RFID technology. Int J Comput Digit Syst 14(1):803–813. https://doi.org/10.12785/ijcds/140162

8. Yuan E, Tong J (2005) Attribute based access control (ABAC) for web services. In: Proceedings of the IEEE international conference on web services (ICWS'05), Orlando, FL, USA. https://doi.org/10.1109/ICWS.2005.25
9. Afshar M, Samet S, Hu T (2017) An attribute-based access control framework for healthcare system. J Phys Conf Ser 933(1):012020. https://doi.org/10.1088/1742-6596/933/1/012020
10. Salehi Shahraki A, Rudolph C, Grobler M (2020) Attribute-based data access control for multi-authority systems. In: 2020 IEEE 19th international conference on trust, security and privacy in computing and communications (TrustCom). IEEE, pp 1834–1841. https://doi.org/10.1109/TrustCom50675.2020.00251
11. Razumov P, Cherckesova L, Revyakina E, Morozov S, Medvedev D, Lobodenko A (2023) Ensuring the security of web applications operating on the basis of the SSL/TLS protocol. E3S Web Conf 402:03028. https://doi.org/10.1051/e3sconf/202340203028
12. Ali AA (2018) Constrained application protocol (CoAP) for the IoT. In: IoT seminar, high integrity system, Frankfurt. https://doi.org/10.13140/RG.2.2.33265.17766
13. Oake A (2021) Diving into RFID protocols with flipper zero. Flipper. https://blog.flipper.net/rfid/
14. Kasagiannis G (2018) Security evaluation of Android Keystore. Bachelor's thesis, University of Piraeus—Department of Digital Systems
15. Rajapaksha D (2021) Integration testing with Spring Boot. Java Code House. https://javacodehouse.com/courses/spring-boot/lesson-7-integration-testing-with-spring-boot/
16. Vadapalli P (2024) Top 8 Raspberry Pi alternatives available in 2024. upGrad. https://www.upgrad.com/blog/raspberry-pi-alternatives/
17. Microsoft (20222) Model-View-ViewModel (MVVM). Microsoft. https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm
18. Onumadu P, Abroshan H (2024) Near-field communication (NFC) cyber threats and mitigation solutions in payment transactions: a review. Sensors 24(23):7423. https://doi.org/10.3390/s24237423