electronics

MDPI

Article

Cloud-Based Access Control Including Time and Location

Mustafa Al Lail [1,*], Marshal Moncivais [1], Robert Benton [2] and Alfredo J. Perez [3,*]

[1] School of Engineering, Texas A&M International University, Laredo, TX 78041, USA; marshalmoncivais@dusty.tamiu.edu
[2] TSYS School of Computer Science, Columbus State University, Columbus, GA 31907, USA; benton_robert@columbusstate.edu
[3] Department of Computer Science, University of Nebraska at Omaha, Omaha, NE 68182, USA
* Correspondence: mustafa.allail@tamiu.edu (M.A.L.); alfredoperez@unomaha.edu (A.J.P.)

Abstract: Location-based services (LBS) offer various functionalities, but ensuring secure access to sensitive user data remains a challenge. Traditional access control methods often need more detail to enforce location-specific restrictions. This paper proposes a new approach that utilizes the Generalized Spatio-Temporal Role-Based Access Control Model (GSTRBAC) within the context of LBS. GSTRBAC grants access based on user credentials, authorized locations, and access times, providing a detailed approach to securing LBS data. We introduce an optimized cloud-based GSTRBAC implementation suitable for deployment in modern LBS architectures. The system uses two secure communication protocols tailored to different security requirements. This allows for efficient communication for less-sensitive data while offering robust protection for highly sensitive information. Additionally, a proof-of-concept mobile application demonstrates the system's functionality and efficiency within an LBS environment. Our evaluation confirms the effectiveness of the cloud-based GSTRBAC implementation in enforcing location-specific access control while maintaining resource and time efficiency.

Keywords: access control; location-based services; authorization; cloud computing

Citation: Al Lail, M.; Moncivais, M.; Benton, R.; Perez, A.J. Cloud-Based Access Control Including Time and Location. *Electronics* **2024**, *13*, 2812. https://doi.org/10.3390/electronics13142812

Academic Editors: Min-Te Sun, Chia-Yu Lin and Hao-Tsung Yang

Received: 27 May 2024
Revised: 7 July 2024
Accepted: 12 July 2024
Published: 17 July 2024

1. Introduction

The rise of cloud computing technology and its seamless integration with the digital world has brought about significant changes in how individuals and organizations conduct business online [1]. Through cloud computing, resources can be stored and accessed digitally without needing physical hardware maintenance. With the recent advancements in cloud technology, numerous cloud providers have emerged to capitalize on this new business opportunity. This has resulted in the development of various services and applications utilizing cloud platforms, including application services (such as MS Office 365, Google Docs, and Google Sheets), data storage services (such as Google Drive and Dropbox), and computing resource services (such as Amazon Web Services, MS Azure, and Google Cloud Platform). Consequently, many organizations have begun outsourcing their data to cloud providers or utilizing their cloud-based applications to optimize workflow.

Cloud technology has come a long way in recent years, but it has vulnerabilities [2]. Researchers have identified several security issues that could be exploited in cloud systems, including physical threats like theft or unauthorized access to resources, as well as communication-based vulnerabilities that allow malicious actors to intercept messages for illicit purposes. As more individuals and organizations rely on cloud systems to remotely access resources through mobile apps, the risk of compromised sensitive information grows. This underscores the importance of adequate security measures to mitigate these issues. Without addressing these vulnerabilities, cloud security breaches could pose a significant threat to society.

Role-Based Access Control (RBAC) [3] and Attribute-Based Access Control (ABAC) [4] are essential access control models that ensure the protection of digital resources by granting

access based on predetermined criteria. These models are employed by various cloud services like Microsoft Azure and Amazon Web Services (AWS) to regulate access to resources within their systems [5,6]. While several researchers have developed customized access control implementations to enhance online security in cloud services, these models need to consider the time and location of access requests [7,8]. For this reason, further research in this area is necessary.

In order to maintain a higher standard for data security, some applications (such as the telemedicine app iMediK [9]) require more than just users' login details to access resources. For example, to ensure that online resources are accessed only in specific spatio-temporal contexts, various constraints can be utilized to limit the conditions in which resources are accessed. This is especially crucial for mobile devices as they can be used in various time frames and can move around easily. Without spatio-temporal controls, these devices risk allowing online resources to be accessed at unintended times or locations.

To illustrate these types of controls, imagine a system (see Figure 1) that uses time and location as additional resource access constraints. In this system, access control policies state that doctors can only access patients' records when they are in the hospital (location constraint) and during their shift (time constraint). If applied to mobile devices that connect to cloud networks, using spatio-temporal controls can reduce the risk of devices being compromised and can prevent access to unauthorized resources. By adding these constraints to the system, it is possible to control the conditions under which resources may be accessed more effectively. Several researchers have developed new models to accommodate spatio-temporal requirements (e.g., Akhuseyinoglu et al. [10], Cao et al. [11], Rantos et al. [12], Xue et al. [13]). In this work, we focus on the Generalized Spatio-Temporal Role-Based Access Control model (GSTRBAC), an extension of RBAC that integrates time and location into authorization decisions [14].

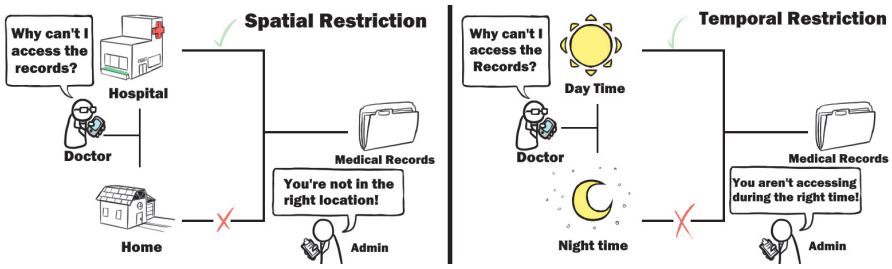

Figure 1. The effect of location and time on resource access.

At present, the GSTRBAC model lacks a cloud-based architecture. To address this, we propose a software architecture for this system. Our proposal outlines how the components should communicate with each other and what tasks they must perform to enable the functionality of the GSTRBAC model within a cloud environment. Furthermore, our paper proposes two security protocols that can encrypt and authenticate messages between the components of the system, ensuring its security. Additionally, we have integrated these protocols within two separate software implementations. We investigate the suitability of these implementations for mobile devices and compare their effectiveness. We evaluate the response time of requests, bandwidth usage, and battery drain of the application to assess the efficacy of these security protocols in practice. In addition, we describe the creation of a web application that showcases how administrators can interact with the database components of the system outside of the mobile application. Through discussing this web application, we highlight our creation of a relational database and a policy database based on UML representations of the GSTRBAC architecture.

This paper makes the following contributions:

1. Proposes a cloud-based software architecture for the GSTRBAC model.
2. Designs and implements two secure communication protocols to encrypt and authenticate messages between the components of the system.

3. Creates two software implementations of the GSTRBAC model based on the proposed secure communication protocols.
4. Performs a comparative analysis of the two security implementations to gauge efficiency compared to each other and a baseline model with no security functions.
5. Creates a relational database and web application based on the GSTRBAC model that allows administrators to interface with the database components of the system.

We organize this work as follows. We provide a review of related work Section 2. In Section 3, we describe an overview of the GSRTBAC model and its purpose in improving access security. Section 4 discusses the creation of a cloud-based software architecture for this model. After this, in Section 5, we examine the security protocols used to ensure secure communication between the components of this system. Section 6 describes the development of a relational database and web-based administration application for the system. Section 7 describes the creation of and the experimentation on the two developed software implementations based on the security protocols established in Section 5. The results of this experiment are presented in Section 8, with follow-up discussion in Section 9. Finally, in Section 10, we discuss the overall results of this work and discuss potential future work for this project.

2. Related Work

Early research in Access Control (AC) systems was conducted during the late 1960s and 1970s with research focused on AC for time-shared systems in multiuser environments [15–18]. With the advent of the Web and the growth of the Internet in the 1990s, along with the development of consumer hardware and software, AC research expanded with the development of Role-Based Access Control (RBAC) [19,20] and other mechanisms [21,22]. The Internet of Things (IoT) pushed further the development of AC methods not only to support logical access for resources such as files, applications, and printers but also to support operations in cyberphysical systems (e.g., industrial control systems) and, later, the inclusion of smartphones and other IoT systems as part of AC management with both decentralized and centralized systems as infrastructure/backends [23–27].

Systems such as IoT devices and smartphones expand the issues that AC systems must work with. These include possible large traffic and data generated by smartphones/IoT devices (there are billions of smartphones and IoT devices [28]), dynamic environments with new devices/users changing, contexts (e.g., environmental conditions), a concern for privacy, and multiple vendors/standards that are expected to be compatible to meet application requirements and use cases [24,27]. The implementation of these AC systems using IoT systems can be undertaken in centralized systems in the cloud [29–33] or by using blockchain technology [32–37].

In the direction of AC systems based on (or using) physical contexts, researchers have identified a growing need for access control models that prioritize the environmental conditions in which system resources are accessed. One aspect that has been studied is the location of users and resources. Consequently, several researchers have proposed access control methods that use location as an additional condition to gain access to resources. The expansion of remote work due to global events (e.g., the COVID-19 pandemic caused by the SARS-CoV-2 virus) has brought interest to access control methods that include location as more workers are working from locations different than their usual offices. For instance, Hu et al.'s Location Aware Semantic Access Control (LASAC) [38] proposes using semantic areas to define where users can access resources. Another example is Xue et al.'s Location-aware Attribute-based Access Control (LABAC) [13], which employs location servers and attribute servers to determine resource access. The RBAC-derived Topology-Aware Access Control (TAAC) model proposed by Cao et al. [11] also considers the location of cyber–physical objects when determining access to resources, in addition to a trust-based risk assessment algorithm.

However, all these proposals overlook the combined integration of temporal and spatial controls. Instead, some researchers have focused on temporal components when

determining resource access. The time at which a user accesses resources is a crucial factor in deciding whether a resource should be accessed or modified. For instance, Hong et al. [39] proposed the Time and Attribute Combined (TAFC) access control to ensure that resources are accessible to users only after specific release windows. Similarly, Balani and Ruj [40] use time intervals to determine resource access. These systems focus on the temporal aspect of a user when determining access to resources, unlike the previous examples that focus on the use of spatial attributes.

Additionally, some projects combine both spatial and temporal elements when determining access to resources. For example, the Extended Generalized Role-based Access Control (EGRBAC) model proposed by Ameer et al. [41] uses environmental factors when making access control decisions in an IoT environment. These environmental factors can vary depending on the situation, like specific times, seasons, or the status of certain objects in the system, to mention a few. However, this project operates within the IoT environment's specific context, making it difficult to use outside of IoT.

In this paper, we focus on the RBAC-derived model GSTRBAC specifically. This model extends the industry-standard access control method of RBAC with spatio-temporal controls. Most available models that use spatio-temporal controls, such as Zhang et al. [42], focus on attribute-based models that more easily integrate spatio-temporal controls into access control decisions. RBAC, on the other hand, is more challenging to integrate spatio-temporal controls into since the base model needs more support for those elements. Hence, RBAC has seen comparatively less research on using spatio-temporal components to influence access control decisions.

3. An Overview of Spatio-Temporal Access Control Model

The contributions of this paper are based on the Generalized Spatio-Temporal Role-based Access Control (GSTRBAC) [14] model. In this section, we describe its components and how this model uses them for access control.

The GSTRBAC model is an extension of the RBAC model that incorporates time and location in access control decisions to resources. The following requirements must be met to access resources under this model:

1. The user must be attempting access within a predefined location.
2. The user must be attempting access during a predefined time interval.
3. The user must possess a role that permits them to access the requested resource(s).
4. The user must have permissions assigned to their role allowing access to the requested resource(s).

The general structure of the GSTRBAC system is illustrated in Figure 2. It consists of five main components, which are Users, Roles, Permissions, STZones, and Spatio-Temporal constraints. STZones are further divided into two sub-components, namely, Time and Location. The main objective of these components is to restrict access to resources based on predefined locations and time intervals. In this system, locations can be classified as physical or logical. Physical locations are specific numerical locations, such as GPS coordinates, while logical locations represent abstract areas, such as Rome, Italy. Time intervals are defined as distinct periods, such as 7:00 a.m. to 4:00 p.m., during which access is valid. The combination of time and location forms discrete STZones which are used in the authorization process to determine access to resources. The exact granularity of these zones (the total area that makes up a location or the length of a given time interval) is an aspect determined by administrators in response to the requirements of any given implementation of this system. For instance, a location can be broad—such as a university campus and all the buildings it encompasses—or it can be narrow—such as a specific building or a section of a building.

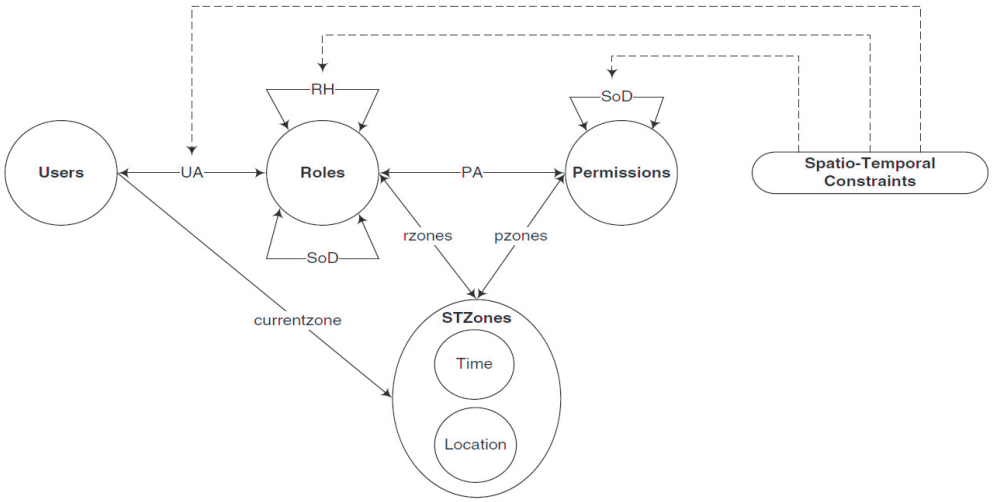

Figure 2. Conceptual model of the Generalized Spatio-temporal Role-based Access Control Model.

Every user in the system is assigned a role that reflects their position in the organization, for example, a manager or a cashier. The role is then assigned specific permissions that dictate which resources can be accessed and what can be done with them. The process of assigning roles to users is denoted as UA, while the process of assigning permissions to roles is denoted as PA.

The relationships between roles and permissions and STZones are indicated by rzones and pzones, respectively. These relationships specify that roles and permissions can only be activated and assigned within certain STZones. To be granted access to resources, the user's current STZone (denoted as currentzone) must match the zones assigned to their role and permission. The connections between these components are depicted in Figure 2.

In Figure 2, there are additional concepts that specify the system's behavior. These concepts include Role Hierarchy (RH) and Separation of Duty (SoD). RH denotes the arrangement of roles within the system relative to each other. For instance, a manager has a more significant set of permissions than an accountant, which means that the accountant's permissions are a subset of the manager's permissions. SoD can be classified into two types: Role SoD and Permissions SoD. Role SoD prevents users from having conflicting roles active simultaneously. For example, it may not be logical for a user to have both a billing clerk role and an accounts receivable role active simultaneously in a particular company. Permissions SoD prevents users from having multiple conflicting permissions assigned to a role simultaneously. For instance, it would not make sense for a user with the role of payroll accountant to have the permissions required to request approval for a payment request while also being able to approve payment requests.

4. GSTRBAC Cloud-Based System Architecture

In this section, we discuss the first contribution of this paper, which is the development of a cloud-based software architecture based on the GSTRBAC model. We will describe the various components of the architecture, including their sub-components and their role in facilitating access control in the GSTRBAC system. Additionally, we will explain how the primary components of the architecture communicate with each other to request access to resources and distribute them to authorized users.

4.1. Overview of the Architecture

The GSTRBAC cloud architecture comprises three primary types of components: a Mobile Application, a Resource Server, and an Authorization Server, which are distributed across several locations (refer to Figure 3). The Mobile Application is the user's point of interaction, allowing them to request access to resources. The resource servers store

resources and grant access to authorized users. On the other hand, the authorization servers hold policy data and make decisions on whether users are allowed to access resources. Furthermore, the server components of this architecture are hosted in the cloud, providing them with excellent storage and computational capabilities.

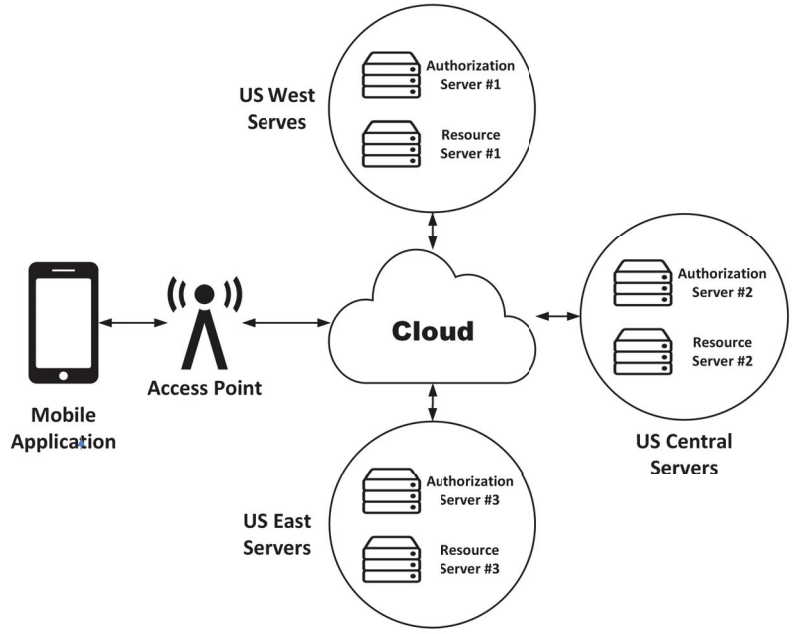

Figure 3. Software architecture of the cloud-based GSTRBAC system.

4.2. Components Communication

The components in the GSTRBAC system are designed to communicate only with their neighboring components. By "neighbors", we mean in an architectural sense, not a physical sense, as seen from the connections in Figure 4. Figure 4 uses the UML component diagram to model the software architecture of the system [43]. The mobile application only communicates with the resource server when making requests since that server is responsible for distributing resources. Similarly, the authorization server only receives messages from the resource server, which is the only component authorized to ask for permission to distribute resources to users. This communication protocol ensures that the system is well-encapsulated, preventing any situation where a user may have direct access to the authorization server, which could potentially allow bad actors to compromise the policy information of the system through the mobile application.

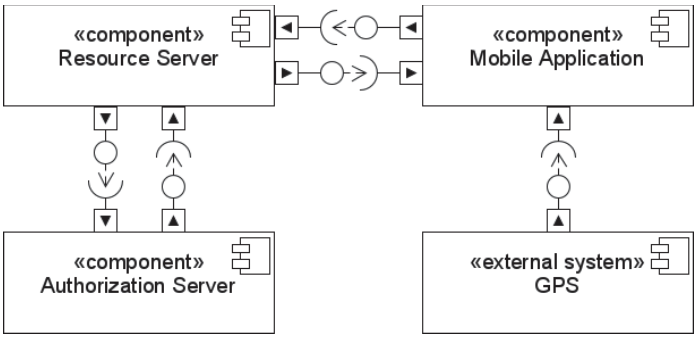

Figure 4. UML Component diagram of GSTRBAC system.

The communication between the different components of the architecture is demonstrated in Figure 5. This process follows a sequence of communication in the system that includes no security features.

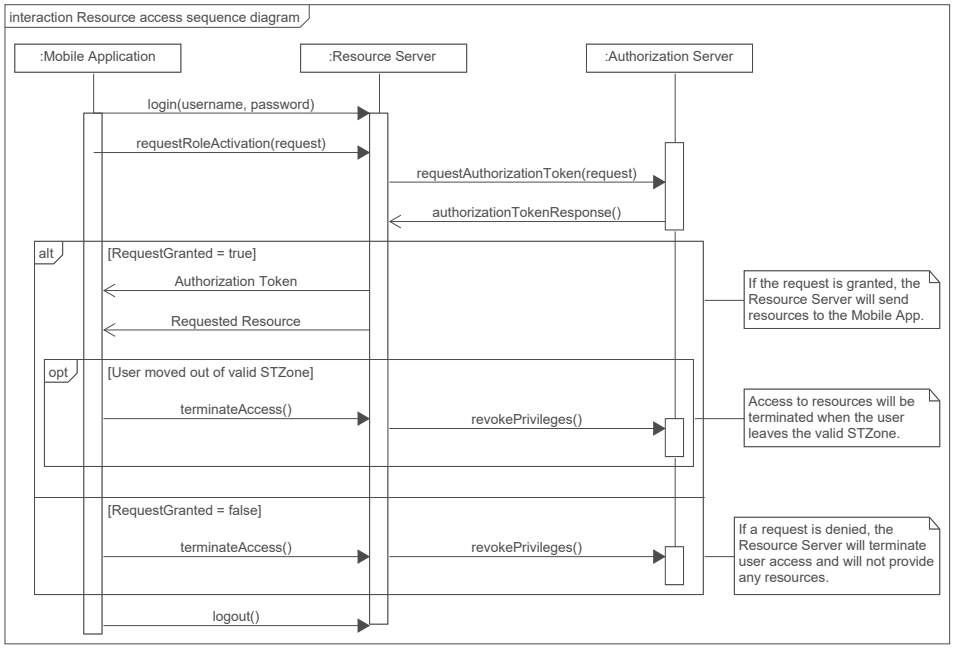

Figure 5. UML sequence diagram for resource access communication protocol.

To begin the communication process, the user must log into the system by sending an authentication request to the resource server. Once the user is authenticated, they can request access to a specific resource. The resource server receives the request and forwards the user's role, STZone data, and the request to the authorization server. The authorization server then sends a message to the resource server with an authorization token if the request is approved. If the request is denied, no token is sent. Two events are possible based on the token request result. If the token is granted, the user can access the requested resource until they disconnect or leave a valid STZone. If the user leaves a valid STZone, access is immediately revoked. This is part of an ongoing authorization process where the mobile application must periodically request access to a resource to maintain access. This ensures that a user, who may initially fulfill the requirements to gain access to a resource, cannot access it later in an unintended context. For instance, if a doctor was granted access to medical records in a hospital during their daytime shift, and they leave the hospital or if their shift ends while accessing those resources, the system detects that the doctor should no longer have access to those records and disconnects them. If the token is not granted, then the user is not granted access to the resource, and they will be asked to enter a valid STZone.

4.3. Mobile Application

The Mobile Application, as shown in Figure 6, serves as the interface between the client and the system. It allows the user to navigate the system using a Graphical User Interface (GUI) and request access to resources. The user selects the desired resource and the action they wish to perform with that resource. The request builder subcomponent constructs the request by collecting relevant user information, such as STZone data and role data, along with the details of the resource and requested action. The zone manager tracks and condenses the user's time and location to provide STZone data to the request builder.

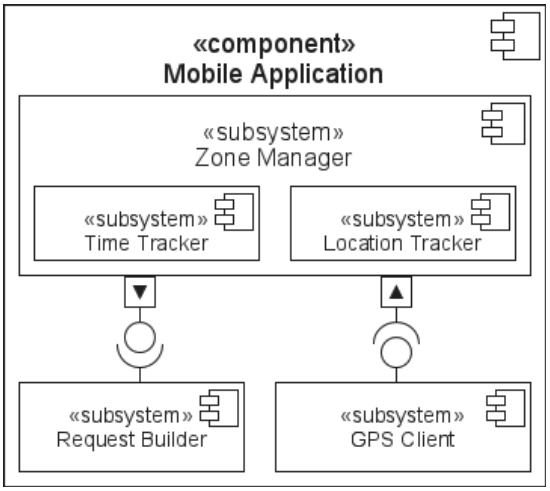

Figure 6. Mobile application software subcomponents.

The request builder sends the request to an available Resource Server and waits for a response. If access is granted, the request builder sends follow-up requests to maintain access to the resource. The system's ongoing authorization process, as discussed in reference to Figure 5, ensures that the user's access to the resource is regularly evaluated and updated.

4.4. Resource Server

The Resource Server, as depicted in Figure 7, is a cloud-based component that stores all the resources in the system in a database. When a user requests a resource, the Resource Server consults the Authorization Server to determine whether the user is authorized to have access to the requested resource. The Resource Server also stores authentication information, such as usernames and passwords, which is used to authenticate users who want to log into the system.

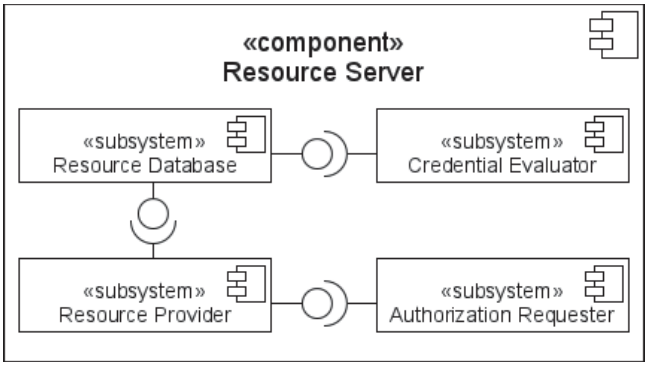

Figure 7. Resource server software subcomponents.

To authenticate the users and determine their authorization, the Resource Server sends a request to the Authorization Server with information such as the user's role, STZone, and the action they wish to perform via the authorization requester subcomponent. If the Authorization Server sends an authorization token, then the relevant resource is collected from the resource provider and sent back to the Mobile Application. If no token is received, then the Resource Server sends a message to the Mobile Application stating that the user does not have access to the requested resource.

4.5. Authorization Server

The Authorization Server, depicted in Figure 8, contains a policy database, also known as the authorization database, that stores all roles and permissions within the system. Whenever a Resource Server makes an authorization request, this request is received by the policy checker subcomponent, which evaluates the information provided by comparing it with the information stored in the authorization database. This information includes the user's STZone data, role, activity ID, and resource ID. If the information provided matches a policy stored in the authorization database, the system marks the user as authorized to access the requested resource. The policy checker then requests a token from the authorization provider and sends a message to the Resource Server that made the authorization request along with this token. However, if the user is not authorized, the policy checker sends a message indicating that the user's information does not allow them to access the requested resource.

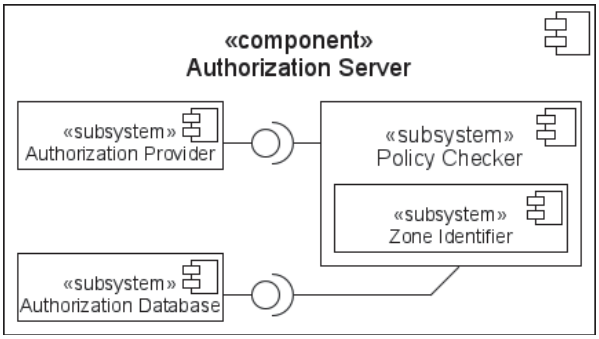

Figure 8. Authorization server software subcomponents.

5. Secure Communication Protocols

Communication between the different parts of the GSTRBAC system is essential for it to work well. However, this also means that messages could be intercepted and data could be compromised while they are being sent. Therefore, it is essential to use security protocols to protect the data stored and sent within the system. This section of the paper is about the second contribution, developing effective security protocols to protect communication between the different parts of the GSTRBAC system. To achieve this goal, the paper proposes two secure communication protocols: a lightweight protocol and a heavyweight protocol.

- Lightweight protocol: This protocol is designed to minimize the impact of cryptography on system resources, taking into account the limited hardware capabilities of mobile devices. It uses cryptography techniques that are known to be less computationally expensive than others.
- Heavyweight protocol. This protocol uses the public-key algorithm RSA to provide the necessary services. RSA is known to be a computationally expensive protocol.

The two security protocols use similar communication methods. They both use digital handshakes to exchange key information, encrypt messages to protect them in transit, use digital signatures to authenticate data, and include non-repudiation. The main difference between the two protocols is that they use different algorithms to protect the information being transmitted. While two versions of the security protocols exist (i.e., lightweight and heavyweight), they are not meant to be dynamically switched. Therefore, a system implementation employing the lightweight protocol will operate equivalently to an implementation using the heavyweight protocol. A sequence of communication that applies to both protocols is shown in Figure 9.

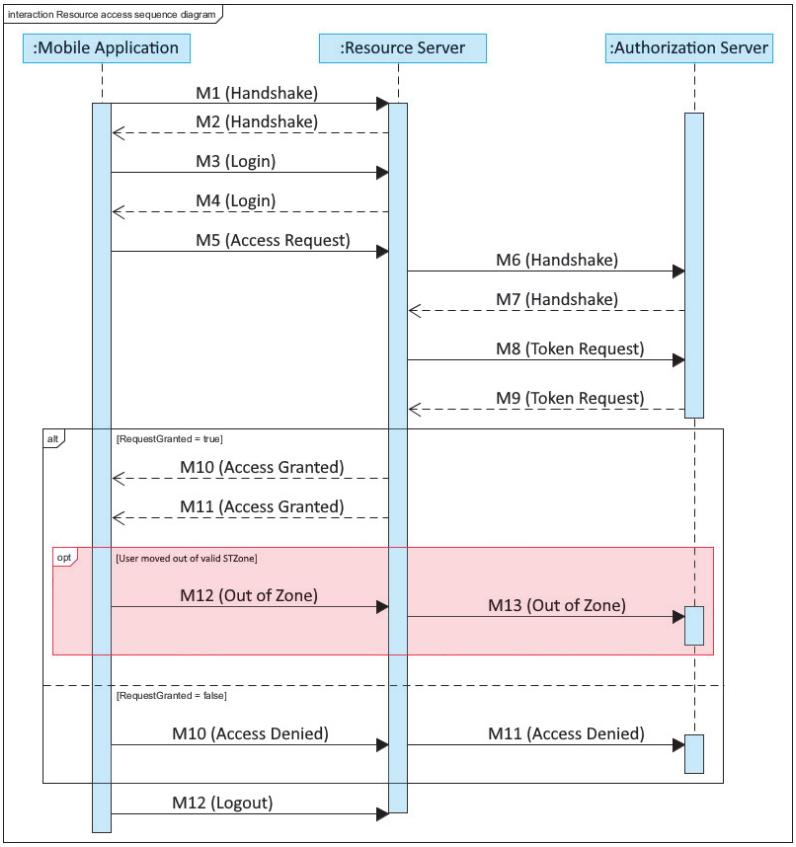

Figure 9. UML sequence diagram of the general security protocol.

5.1. Lightweight Protocol

Because system resources are limited, it is preferable for any security protocols used to conserve resource use whenever possible. The lightweight communication protocol accomplishes this by prioritizing computational speed and the conservation of system resources. The use of elliptic curve algorithms and other lightweight algorithms ensures the system has a quick, secure, and computationally cost-effective method of computation that saves power over time. This algorithm's handshake is driven by the Elliptic Curve Diffie–Hellman (ECDH) which allows for a secure creation of a shared key for encryption. This key is used to encrypt and decrypt all messages between two components until communication is fully concluded between them. This protocol uses the Advanced Encryption Standard (AES) to encrypt and decrypt these messages. Finally, this protocol uses the Elliptic Curve Digital Signature Algorithm (ECDSA) to create digital signatures for authenticating messages. Additionally, this protocol is a symmetric form of encryption which means that the use of the digital signatures will be more complicated and may require a separate key pair for proper use.

5.2. Heavyweight Protocol

Some systems may wish to use a more resilient security protocol even at the cost of higher system resource usage. In response to this, we have designed the heavyweight protocol to be simpler and more resilient than the lightweight protocol in terms of its structure while also being more computationally expensive. This is because the heavyweight protocol uses Rivest, Shamir, and Adleman (RSA) for the handshake, encryption, and digital signature functions. By using only RSA for all of these communication functions rather than a combination of multiple different algorithms, the system can be viewed as theoretically simpler while still being robust in terms of the protection offered. Additionally, because this protocol is an asymmetric model, this means that it is capable of handling all its

encryption protocols using a single algorithm since the private and public keys can be used to provide the functions of confidentiality or authentication depending on the situation.

6. Web-Based Administration Application

This section presents our fifth contribution for this paper, the creation of a relational database that provides a way to store and retrieve GSTRBAC policy information. This database is accessed through a web application that security analysts can use to administer GSTRBAC policies. This relational database and the web application highlight the method by which we can take the abstract model of the GSTRBAC database and create a means for administrators to modify policy information within the GSTRBAC system. To a degree, this also represents our third contribution to this paper (the software implementations of the GSTRBAC model) since it helps contextualize the administrative environment of the system.

The GSTRBAC model was designed using the class models of the Unified Modeling Language (UML), the standard modeling language in the software industry [43]. Figure 10 depicts the UML class model representing the different components of the GSTRBAC model and their interrelationships with time and location. We developed this class model by examining the conceptual model shown in Figure 2 and identifying the critical nouns used to describe the model. These nouns are translated into classes that can be more tangibly manipulated. These classes are assigned attributes that facilitate their purpose, and relationships are drawn between classes that interact with each other in specific ways. The diagram in Figure 10 shows how spatio-temporal elements interact with various elements of the GSTRBAC system such as role assignment or permission activation. The various classes within this system have different relationships with other classes that help facilitate access control using both roles and spatio-temporal attributes. However, the UML class model only provides an abstract representation of GSTRBAC policies and lacks descriptions of how these policies should be stored, retrieved, and administered.

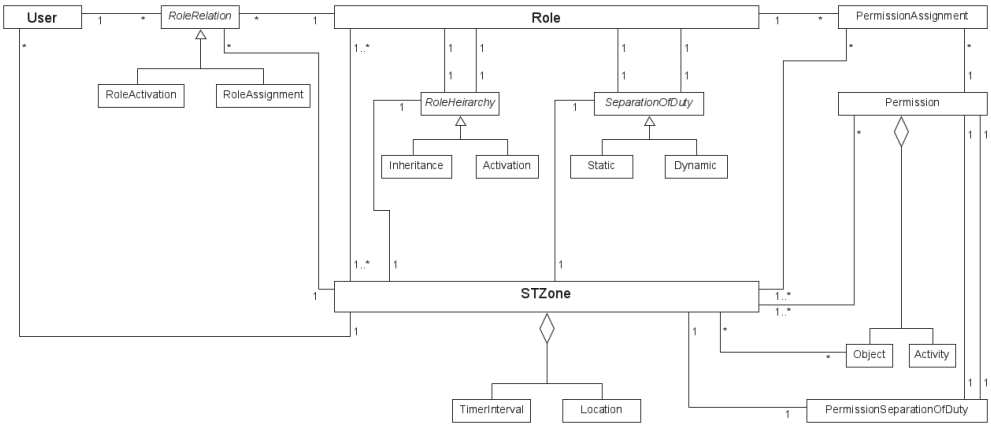

Figure 10. UML class model representing GSTRBAC policies.

To resolve this, we first discuss how we convert the UML policy class model to a relational database schema, which is then used to create policy databases. These databases are then used to store and administer GSTRBAC policies. Then, we use the Structured Query Language (SQL), C# programming language, and the Visual Studio Table Designer tool to define, manipulate, and access a GSTRBAC policy database for a simple company system. Finally, we outline a web application (developed using the ASP.NET Model View Controller (MVC) framework) that can be used by access control analysts for the administration of the policy database.

6.1. GSTRBAC Relational Database

In order to store the policy information of the GSTRBAC system, it is necessary to translate the UML class model (Figure 10) into a more explicitly defined relational database. The result of this process is the relational database shown in Figure 11 which explicitly defines what information should be stored in each component of the system. We apply the following set of rules that convert the UML class model to the relational database schema, as suggested by Alvaro Monge [44]:

1. Map UML classes to tables in the relational database schema.
2. MAP attributes of a UML class to fields of corresponding tables.
3. Create primary keys for all tables, if not already included in fields.
4. One-to-many associations between classes in UML are represented as foreign keys in the schema diagram.
5. Many-to-many associations between UML classes are represented as a separate table that connect the classes of the associations together.

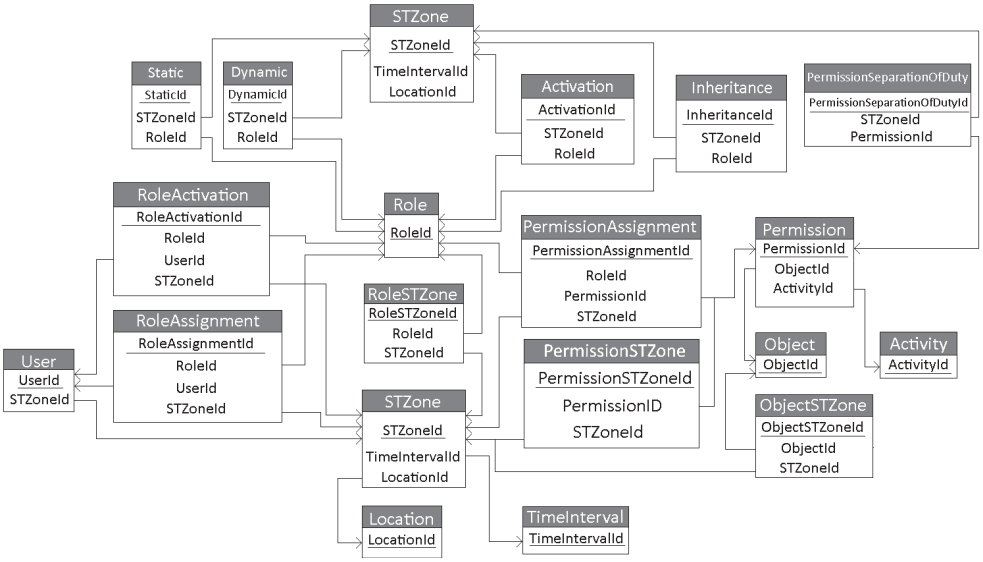

Figure 11. The relational database schema produced from the conversion of GSTRBAC UML class model in Figure 10.

Applying these rules, we can convert all classes in Figure 10 to schema tables. In the conversion process, we keep the names of UML classes (indicated by the gray boxs in the diagram) the same after translating them to tables. After converting the classes to tables, any attributes of the classes are assigned as fields to the corresponding named tables.

In the GSTRBAC UML class model (Figure 10) RoleHierarchy and SeparationOfDuty represent generalization–specialization relationships. We convert these UML classes to schema tables by using the foreign keys of the generalizations in the specialization classes. This means that if a generalized class has access to certain foreign keys, the specialization classes associated with it will also have these foreign keys. For example, SeperationOfDuty has two specialization classes: Static and Dynamic. The generalization class has a one-to-many association with the STZone and Role classes; hence, two foreign keys are used.

6.2. Database Implementation and Administration

To demonstrate the applicability of GSTRBAC policies, we instantiated the policy schema in Figure 11 using a simple company as an example. To do this, we used Visual Studio's table designer to create the company policy database. To interact with the policy database, we used the ASP .NET Model View Controller (MVC) application, a tool used by security analysts for administration purposes.

Policy Database Development

When implementing the policy database, we use SQL as the database definition and manipulation language. Using Visual Studio, we use the Table Designer tool included in the SQL Server database development package to create and instantiate databases, including the policy database and the tables for the schema shown in Figure 11.

After creating the database tables, we instantiate them with data modeled after a software development company. We represent these data as datasets encapsulating the overall concepts of the GSTRBAC model. These datasets represent information that a potential company may allow administrators to manage. Because the web application requires a model of the database to function, it is necessary to explicitly clarify the different data types used by the model. This ensures that the data used by the web application correctly correspond with the types outlined in the original model. This means that information provided regarding a company must be aligned with the appropriate data types representative of that information. After the tables were instantiated with these data, we began the development of the external web-based application using these new tables as a basis. We follow this process as the web application requires a model of the database. For this reason, it is necessary to clarify the different data types used by the model so that they correctly match the database. This will ensure that the data used by the web application will correctly correspond with the types outlined in the original model.

6.3. Administration Web Application

ASP.NET MVC is a framework for developing web applications in Visual Studio using the C# language [45]. This framework enables the communication between a database and a web application using the Models, Views, and Controllers architecture (MVC). Models represent the data in the system and the state of the application itself. Views are responsible for presenting content to users through the GUI. Finally, Controllers handle user interaction, manipulate data from the Model, and select a view to render to the user. We use the different elements of the MVC framework to form a comprehensive application that allows administrators to interact with the GSTRBAC system.

In the MVC framework, Models represent the structure of a database using classes in C#. The other two components of the MVC pattern use Models as a source of data necessary both to conduct program functions and to display visual information to users. For this reason, it is necessary to ensure that these data are stored in a clear and readable format. Initially, this is achieved by denoting the type of data that should be retrieved or stored in the database. The View component then displays data from a table found in the database to a web application following the structure of the Model. It does so in conjunction with the Controller and Model components. This way, the Controller passes data to the View component where we format them visually to allow for easier administration. Further, Controllers provide the logic for manipulating data and passing them to the View to display. In the Controller component, we develop the complex actions that administrators then use in policy management.

In order to fully describe the development of the web application using the MVC framework, it is necessary to give context by describing Language-Integrated Queries (LINQ), Razor, and HyperText Markup Language (HTML). LINQ is a variant of SQL that C# programs use to query the data from a database and is the primary language used in the Controller component of MVC. Razor and HTML are coding formats used in the development of web pages. HTML is the standard language used in this aspect and is used to determine how a web page will be displayed. For the web application of this paper, the View component uses HTML to display administrative information. Razor acts similarly to HTML and compresses HTML to allow for improved readability and management. Razor is used in the View component to allow for improved communication between the View and Controller components.

The directory page of this web application allows for the manipulation of data in the tables created previously. This includes the various GSTRBAC elements such as users, roles, locations, objects, etc. The web application displays these pages through the MVC View component, which enables administrators to pass data to Controllers. The Controllers then use the input in LINQ queries to perform actions predefined by the company's policies. Through this interface, administrators can handle tables and perform maintenance on the policy database through GUI elements. Although this system does not directly interface with the final software implementation developed based on the architecture, it demonstrates the method by which administrators can use a relational database to store policy data and influence how access is distributed in the system.

7. Implementation and Experimentation of the System

The third contribution of this paper is the development of multiple software implementations incorporating both the software architecture proposed in Section 4 and the security protocols proposed in Section 5. Within this section, we discuss what these implementations contain and how they implement the concepts outlined in Sections 4 and 5. Additionally, this section discusses our fourth contribution, the experiments conducted to evaluate the performance of these implementations. We first describe the baseline implementation which features no security protocols and exists as the foundational implementation that demonstrates the functionality of the cloud-based GSTRBAC system. We then discuss the lightweight and heavyweight protocol implementations which build upon the design of the baseline implementation through the use of the associated security protocols. After this, we discuss the experiments conducted on these implementations to gauge system resource usage and general efficiency compared to one another.

7.1. Cloud-Based Implementation

The baseline implementation incorporates the three main components of the architecture described in Section 4. This implementation contains a user application that operates as the Mobile Application component and cloud servers hosted to house the Resource Servers and Authorization Servers.

The Mobile Application allows users to log into the system and then make resource requests as discussed in Section 4. The GUI demonstrating the user experience for this process is shown in Figure 12. Through this GUI, users first must log into the system and be successfully authenticated by it. After logging in, users are presented with a resource request screen. This screen provides options for selecting a resource to access, the desired action to use in conjunction with the resource, and the user's role in the system. Depending on these factors, in addition to the time and location of access, the user will either be granted access to the requested resource or will receive an in-app notification that they are not authorized to access the requested resource and will be sent back to the resource request screen. Additionally, if granted access, users will need to maintain their STZone, or the system will inform them that they are no longer in a valid STZone and it will return them to the resource request screen.

The Resource and Authorization Servers are hosted on cloud-based virtual machines. For this paper, we host the servers using Microsoft Azure's cloud services, though the current implementation is not designed to be provider-specific and should be applicable to any cloud provider. These servers are Standard_B2s VMs which use 2 vCPUs, 4 GB of memory, and 8 GB of SSD storage. These servers communicate with each other and the Mobile Application over the Internet, with the Resource Servers and Authorization Servers hosted on separate machines to ensure proper encapsulation of data. This is done to prevent multiple components from being compromised at once in the event of a security breach and to ensure that users cannot directly connect to the Authorization Server, potentially compromising the stored policy data.

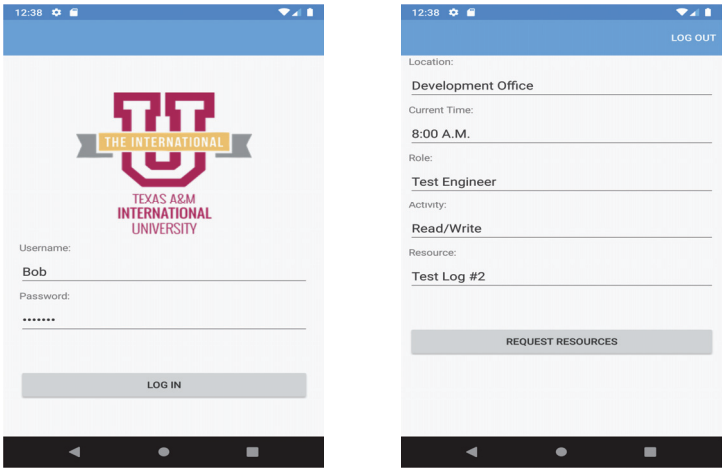

Figure 12. Android mobile application user-interface.

The server components of this system are hosted in multiple areas of the United States. Since there are two components that must be housed in servers, we placed two servers in three different time zones of the country, totaling six servers. This was done to improve the response time of these servers when a resource request is made. Because users may be mobile and can attempt access from different parts of the country, having multiple options for accessing resources may be useful for ensuring constant access to system resources. This is in addition to acting as a security fail-safe featuring multiple points of redundancy. Using this arrangement, if an authorization server in one part of the country is disabled, it is still possible to request authorization to access resources. Additionally, if a Resource Server is disabled, users can still access requested resources from any other available server. For the purposes of this paper, a method of synchronizing data between these servers is assumed, with the precise method being outside its scope.

The lightweight and heavyweight implementations of the system are built upon the baseline implementation, reusing most of the same code and structure. Visually, there is no noticeable difference in the operation of these versions of the GSTRBAC system and the baseline version except for a longer wait time between access requests. This is because the main difference between these implementations is in how communication between the components is handled. The lightweight and heavyweight implementations use their respective security protocols to encrypt messages in transit, exchange key information, and create digital signatures as described in Section 5. This is different from the baseline implementation which sends messages without any form of encryption or message authentication. The presence of these protocols means that the time between a user making a request and receiving access to a requested resource is longer. This extended length, however, is a necessary drawback so long as the messages exchanged by this system are better secured and the longer length of resource requests is not too extensive.

7.2. Experiment Setup

To determine the efficacy of these security implementations in regard to system performance, we designed an experiment to gauge their impact on system resources over a prolonged period. By doing this, we can compare any potential strengths and weaknesses between the different implementations while also seeing how they perform in relation to the unsecured baseline implementation.

In this experiment, we compared three implementations of the GSTRBAC system:

1. Implementation A: A version with no security when transferring data (Baseline).
2. Implementation B: A version with lightweight security. The lightweight security implementation utilizes a combination of secret key encryption and digital signatures (ECDHA, AES, ECDSA).

3. Implementation C: A version with heavyweight security. The heavyweight security implementation, on the other hand, uses public-key cryptography to safeguard data (RSA).

The factors we examined in this experiment are the power consumption, response time, and the amount of data transferred by each implementation. By examining these specific elements, we can determine which areas a given implementation may be stronger or weaker in compared to the other versions. Our expectations when creating this experiment were that the heavyweight security options would require more power and would take longer to fully send and receive messages. This increased cost, however, would be acceptable so long as the benefits the protocol provides sufficiently outweigh the drawbacks. The question in regard to the heavyweight implementation is whether or not the presumed increase in power consumption, response time delay, and possibly the amount of data transferred would make the program more efficient to use for a prolonged period of time purely for the benefit of using a more rigorous encryption algorithm. And in the case that it was less efficient to use, by how much was usage made less efficient?

The experiments were designed to run for eight hours each and were performed multiple times for each implementation. These implementations were designed to operate on Android devices, with physical devices used to conduct these tests. Each test had the user log into the system and request a resource. After requesting a resource, the system's process of ongoing authorization would continually check to make sure that the user was still allowed to access the requested resource. This process would continue for the duration of the experiment until the experiment's run-time concluded or the user's battery died, whichever came first. After the conclusion of the experiment, we analyzed the data on the three measured factors and compared them to the factors of the other implementations.

As an additional note regarding the duration of these experiments, the eight-hour duration was chosen so that the application could run long enough to obtain the necessary data points while also being short enough to conduct the number of tests needed within a reasonable time frame. Originally, we also attempted to record the duration of the experiments as a fourth data point under the assumption that the program may drain the battery life of the test phones being used before the full duration of the test could be completed. By examining the length of time the Mobile Application could be run before fully draining the battery, it would be possible to determine whether or not a given implementation is viable for practical use since battery usage is an important consideration for the limited hardware of mobile devices. However, recording the length of time the experiments ran for proved to be unnecessary. There were no cases in which the phones ran out of battery, and nearly all experiments ran for the full duration of the test without error. The tests that did conclude earlier than the eight-hour run time did so for reasons unrelated to any battery drain.

7.3. Power Consumption

Examining the battery life of the user's device as they used the various implementations was necessary to determine how application usage could be impacted in the long term. If the battery cannot sustain prolonged usage of the GSTRBAC implementation, then there may be issues regarding the practicality of the system itself. The battery usage of the device must be reasonable for the actions carried out by the system, preventing unnecessary monopolization of system resources to accomplish those actions. For this reason, we examine the battery usage of the system so we can determine what optimizations need to be made to reduce the impact on battery life, assuming any optimizations can be made.

For gauging power consumption during the experiments, an application called Battery-Historian is used to display the battery usage statistics of users of the system. This program provides detailed information regarding the battery life over a period of time as well as usage statistics of individual programs on the user's Android phone. Using this data, we could then see how battery life was affected depending on the system implementation used.

7.4. Response Time

Response time refers to how long it takes the user to receive information from the Resource Server component after making an access request. Ideally, the delay between the user requesting access to resources and receiving (or being denied) access should be short. However, because the Resource Server is not on the same machine as the client's device, there is an element of latency that must be accounted for. This is compounded by the need for access requests to be authorized by the Authorization Server before resources can be distributed. Additionally, the processing time for the user's request may differ depending on how many zones and permissions are defined within the system, the quantity of data being requested by the user, or how far away the data centers housing the servers are from the user. For example, if a user in California wished to request data from the GSTRBAC system but the only available Resource Server was located in New York, then the request would take longer than if the server was also located in California.

The response time in the experiment is determined by an internal timer defined within the system that notes when the access request is made and when a response is received. The response time data are measured on the mobile device during run time and are output to a text file where they are collected at the conclusion of the experiment. There are two instances in which the response time is recorded. The first is when the user logs into the system and needs to be authenticated. This response time is expected to be relatively short since only the Resource Server is contacted to perform authentication. The second time the response time is measured is when users request resources from the Resource Server. This instance is longer since both the Resource Server and Authorization Server must be contacted to fulfill the resource request. Both of the response times collected during these instances are useful for determining potential bottlenecks in terms of system performance or connectivity issues.

7.5. Amount of Data Transferred

We examined the amount of data transferred in the system to see how much overhead is generated by the use of security. It is reasonable to assume that both security implementations will transfer a greater amount of data than the baseline implementation. This is due to the additional information overhead required to accommodate encrypted messages and the digital signature that is necessary to authenticate those messages. Although the additional overhead is necessary to have access to functional security protocols, it is useful to set expectations regarding how much data will be transferred through the system on average.

We collect information regarding data transfer amounts through the BatteryHistorian application. Although BatteryHistorian is mainly used to determine battery use statistics, it also provides application-specific information on data that are sent and received by a cell phone. This allows us to easily determine the quantity of data sent and received by the application while it was running for the test.

8. Results

We can note specific differences in the results of all three implementations, all of which are shown in Table 1. The baseline implementation by default displayed the fastest and least-impactful results (in terms of affecting device performance). We use these baseline numbers to determine the level of impact the security protocols had on the general performance of the system. The lightweight implementation showed an increase in overall resource usage compared to the baseline. This includes using slightly more power overall, having a longer response time, and having a higher average of total data transferred. This is to be expected since the security measures implemented would require additional resources to operate. The heavyweight implementation, by comparison, showed an even higher average power usage and response time but a lower amount of data transferred. For comparison's sake, the lightweight implementation has a data transfer overhead of roughly 86% compared to the baseline while the heavyweight implementation only has an overhead of roughly 69%. From these data, we can say that the heavyweight is more costly power-wise and time-wise

than the lightweight but is marginally more data-efficient. We may be able to attribute this to RSA encrypting messages and creating digital signatures more efficiently, albeit at the cost of additional time and energy.

Table 1. Experiment results.

Security Protocol	Avg. Response Time (ms)	95% Confidence Interval (ms)	Avg. Data Transferred	Avg. Power Consumption	Data Overhead	Power Consumption Overhead
Baseline	292.64	223–337	7.45 MB	11.63%	N/A	N/A
Lightweight	1873.05	1864.89–1872.45	13.9 MB	12.52%	86.58%	7.65%
Heavyweight	2160.32	2152.67–2167.97	12.59 MB	12.93%	69%	11.12%

The 95% confidence intervals shown in Table 1 highlight the expected range of response time values for a given protocol. These values can be interpreted as a representation of the latency distributions for the protocols in this table. Generally, the range of values for the implementations utilizing the security protocols had a low internal interval of around 10–15 ms but differed from one another by around 300 ms. The performance impact of the heavyweight implementation of about 15% could indicate non-negligible performance costs, assuming the cost displayed here is maintained for larger message sizes.

Experiment Design Limitations

The collection of data for these tests mainly focused on the mobile application portion of the GSTRBAC system implementations. This is due to the lack of a method of isolating the resource usage of our server components. However, for collecting data about the mobile component, we used BatteryHistorian to gain data regarding the various statistics we examined in Table 1. Additionally, we were limited to collecting these data points from the mobile device since we could not isolate this usage through our system data collection software. The amount of data we could collect was further limited by resource constraints during testing which limited the number of devices we could test at once.

In terms of experimental error, there were two instances where the program failed to run for the full eight-hour duration. However, extrapolating the data for those cases to the full eight-hour run-time shows values consistent with the results of experiments that did run to completion successfully. It is unclear why these tests specifically failed, but it is possible that interruptions in Internet traffic may have caused a lapse in the connection of the device. If the Mobile Application loses connection to the Resource Server for a long enough period, it sends the user back to the resource request screen. Because the tests were eight hours long, we could not reasonably monitor the devices directly at all times and reconnect to the Resource Server in a timely manner. If we attempted to reconnect to the server after too much time elapsed we would risk distorting other values of the test, potentially invalidating them.

9. Discussion

This model is focused on ensuring that resources are used within their proper contexts. In that respect, the spatio-temporal controls introduced by this model allow for the control necessary to ensure that resources are only being accessed by people in pre-approved contexts. This system, however, relies on proper configuration by administrators to work effectively. To ensure the controls introduced by this system are secure, administrators must work to ensure that users are being assigned the correct roles to access resources within any system employing this model. How and when roles are assigned to users must be determined on a case-by-case basis to accommodate the usage of the model by administrators. For example, this system may run into issues if used for a system with constantly changing roles. In this instance, the methods by which resources are restricted will remain constant. In a dynamic system, however, the particulars of resource access

become more complicated and can introduce considerable overheads when administrators must consider what roles should access which resources, who should be assigned which roles, and under what spatio-temporal contexts access should be granted. In this situation, it may be necessary to create broader roles that allow for broader requirements for access rather than narrow ones to reduce administrative overhead. The amount of easing that would be acceptable would, of course, need to be determined by administrators using this system.

These administrative decisions, while important in determining the effectiveness of any given system, are not within the scope of this paper, which is focused on the performance of a system implementation of GSTRBAC. For this reason, we do not examine data processing or how data are shared with third parties. These elements must be decided on a case-by-case basis according to the needs of administrators and users. The results shown in our testing show response times connected to small and repeated data transfers. The average response time would naturally worsen for access requests with larger messages or messages with additional data processing attached. These are concerns, alongside the privacy of users, that must be considered by administrators when determining how to handle their data within this system.

10. Conclusions

The increased prevalence of cloud services for personal use, commercial and industrial enterprises, and government management of resources has led to a greater need for stronger authorization controls to manage these services. This paper has focused on the development of a cloud-based software architecture using the GSTRBAC model to offer added spatio-temporal controls to cloud systems. By using these spatio-temporal controls, administrators can better control where and when resources are accessed and distributed in cloud systems. We also developed two security protocols to provide additional protection for messaging between the different components of this system. This includes the addition of a handshake protocol to share key information, message encryption, and message authentication through the use of digital signatures. In addition to this, we also developed multiple software implementations of this system. One implementation focused on the use of a relational database and the creation of a web application to manage policies within the GSTRBAC system. Three other implementations demonstrate how the system components of the GSTRBAC system communicate to determine access to resources and subsequently deliver them if access is granted. We tested these implementations which included both the lightweight and heavyweight security protocols as separate implementations, as well as a baseline implementation for comparison. At the conclusion of our tests, we found that the lightweight model outperformed the heavyweight in terms of power usage and response times for handling resource requests, while the heavyweight implementation outperformed the lightweight implementation in terms of the amount of data transferred.

For future work, we would like to conduct a more expansive study using a larger sample size. This would allow us to determine the scalability of this system as opposed to this current implementation, which was more focused on a narrower subset of this kind of system. While conducting these tests, we could determine how a larger implementation of this system could respond to current threats and vulnerabilities and determine how this system can be made more adaptable, both in terms of how this system can integrate with existing software systems and in terms of how the system can respond to the appearance of new threats and vulnerabilities in the future.

Author Contributions: Conceptualization, M.A.L.; Software, M.M. and R.B.; Investigation, M.A.L., A.J.P., M.M. and R.B.; Resources, M.A.L. and A.J.P.; Writing—original draft, M.A.L. and M.M.; Writing—review and editing, M.A.L. and A.J.P.; Supervision, M.A.L. and A.J.P.; Project administration, M.A.L. and A.J.P.; Funding acquisition, M.A.L. and A.J.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work received partial support from the U.S National Science Foundation grant awards 1950416 and 2308741, and the Presidential Research Development grant from Texas A&M International University.

Data Availability Statement: The full datasets created while conducting our experiments can be located at https://github.com/MarshMonci/GSTRBAC (accessed on 6 July 2024).

Acknowledgments: We thank the anonymous reviewers who have helped us to improve the content, organization, and presentation of this work. The authors also want to acknowledge Carlos Delgado who helped with software implementation details.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

AC	Access Control
RBAC	Role-based Access Control
GSTRBAC	Generalized Spatio-temporal Role-based Access Control
STZones	Spatio-temporal zones
UA	User Assignment
PA	Permission Assignment
RH	Role Hierarchy
SoD	Separation of Duty
UML	Unified Modeling Language
GUI	Graphical User Interface
MVC	Models, Views, and Controllers architecture
SQL	Structured Query Language

References

1. Gouglidis, A.; Mavridis, I.; Hu, V.C. Security policy verification for multi-domains in cloud systems. *Int. J. Inf. Secur.* **2014**, *13*, 97–111. [CrossRef]
2. Ahmed, M.; Litchfield, A.T. Taxonomy for Identification of Security Issues in Cloud Computing Environments. *J. Comput. Inf. Syst.* **2018**, *58*, 79–88. [CrossRef]
3. Sandhu, R.S.; Coyne, E.J.; Feinstein, H.L.; Youman, C.E. Role-based access control models. *Computer* **1996**, *29*, 38–47. [CrossRef]
4. Hu, V.C.; Kuhn, D.R.; Ferraiolo, D.F.; Voas, J. Attribute-based access control. *Computer* **2015**, *48*, 85–88. [CrossRef]
5. Microsoft. What Is Azure Role-Based Access Control (Azure RBAC)? 2024. Available online: https://learn.microsoft.com/en-us/azure/role-based-access-control/overview (accessed on 6 July 2024).
6. Amazon. Using Role-Based Access Control. 2024. Available online: https://docs.aws.amazon.com/cognito/latest/developerguide/role-based-access-control.html (accessed on 6 July 2024).
7. Zhu, Y.; Hu, H.; Ahn, G.J.; Huang, D.; Wang, S. Towards temporal access control in cloud computing. In Proceedings of the 2012 Proceedings IEEE INFOCOM, Orlando, FL, USA, 25–30 March 2012; pp. 2576–2580. [CrossRef]
8. Yu, S.; Wang, C.; Ren, K.; Lou, W. Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. In Proceedings of the 2010 Proceedings IEEE INFOCOM, San Diego, CA, USA, 14–19 March 2010; pp. 1–9. [CrossRef]
9. Maji, A.K.; Mukhoty, A.; Majumdar, A.K.; Mukhopadhyay, J.; Sural, S.; Paul, S.; Majumdar, B. Security analysis and implementation of web-based telemedicine services with a four-tier architecture. In Proceedings of the 2008 Second International Conference on Pervasive Computing Technologies for Healthcare, Tampere, Finland, 30 January–1 February 2008; IEEE: Piscataway, NJ, USA, 2008; pp. 46–54.
10. Akhuseyinoglu, N.B.; Joshi, J. A risk-aware access control framework for cyber-physical systems. In Proceedings of the 2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC), San Jose, CA, USA, 15–17 October 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 349–358.
11. Cao, Y.; Huang, Z.; Yu, Y.; Ke, C.; Wang, Z. A topology and risk-aware access control framework for cyber-physical space. *Front. Comput. Sci.* **2020**, *14*, 144805. [CrossRef]
12. Rantos, K.; Fysarakis, K.; Manifavas, C.; Askoxylakis, I.G. Policy-Controlled Authenticated Access to LLN-Connected Healthcare Resources. *IEEE Syst. J.* **2018**, *12*, 92–102. [CrossRef]
13. Xue, Y.; Hong, J.; Li, W.; Xue, K.; Hong, P. LABAC: A location-aware attribute-based access control scheme for cloud storage. In Proceedings of the 2016 IEEE Global Communications Conference (GLOBECOM), Washington, DC, USA, 4–8 December 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–6.

14. Abdunabi, R.; Al-Lail, M.; Ray, I.; France, R.B. Specification, validation, and enforcement of a generalized spatio-temporal role-based access control model. *IEEE Syst. J.* **2013**, *7*, 501–515. [CrossRef]
15. Fano, R.M.; Corbató, F.J. Time-sharing on computers. *Sci. Am.* **1966**, *215*, 128–143. [CrossRef]
16. Graham, G.S.; Denning, P.J. Protection: Principles and practice. In Proceedings of the Spring Joint Computer Conference, Atlantic City, NJ, USA, 16–18 May 1972; pp. 417–429.
17. Lampson, B.W. Protection. *ACM SIGOPS Oper. Syst. Rev.* **1974**, *8*, 18–24. [CrossRef]
18. Saltzer, J.H. Protection and the control of information sharing in Multics. *Commun. ACM* **1974**, *17*, 388–402. [CrossRef]
19. Sandhu, R.S. Role-based access control. In *Advances in Computers*; Elsevier: Amsterdam, The Netherlands, 1998; Volume 46, pp. 237–286.
20. Abdi, A.I.; Eassa, F.E.; Jambi, K.; Almarhabi, K.; Al-Ghamdi, A.S.A.M. Blockchain platforms and access control classification for IoT systems. *Symmetry* **2020**, *12*, 1663. [CrossRef]
21. Qiu, J.; Tian, Z.; Du, C.; Zuo, Q.; Su, S.; Fang, B. A survey on access control in the age of internet of things. *IEEE Internet Things J.* **2020**, *7*, 4682–4696. [CrossRef]
22. Di Francesco Maesa, D.; Mori, P.; Ricci, L. Blockchain based access control. In Proceedings of the Distributed Applications and Interoperable Systems: 17th IFIP WG 6.1 International Conference, DAIS 2017, Held as Part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, Neuchâtel, Switzerland, 19–22 June 2017; Proceedings 17; Springer: Berlin/Heidelberg, Germany, 2017; pp. 206–220.
23. Das, S.; Saraf, C.; Khairnar, D.P. A Hyperledger Fabric Based Organizational Decentralized Access Control Solution. In Proceedings of the 2020 IEEE 7th International Conference on Engineering Technologies and Applied Sciences (ICETAS), Kuala Lumpur, Malaysia, 18–20 December 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 1–6.
24. Lee, Y.; Lee, K.M. Blockchain-based RBAC for user authentication with anonymity. In Proceedings of the 2019 ACM Conference on Research in Adaptive and Convergent Systems, Chongqing, China, 24–27 September 2019; pp. 289–294.
25. Alshehri, A.; Sandhu, R. Access control models for cloud-enabled internet of things: A proposed architecture and research agenda. In Proceedings of the 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), Pittsburgh, PA, USA, 1–3 November 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 530–538.
26. Bhatt, S.; Patwa, F.; Sandhu, R. An access control framework for cloud-enabled wearable internet of things. In Proceedings of the 2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC), San Jose, CA, USA, 15–17 October 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 328–338.
27. Gupta, M.; Bhatt, S.; Alshehri, A.H.; Sandhu, R. *Access Control Models and Architectures for IoT and Cyber Physical Systems*; Springer: Berlin/Heidelberg, Germany, 2022.
28. Perez, A.J.; Zeadally, S. Recent advances in wearable sensing technologies. *Sensors* **2021**, *21*, 6828. [CrossRef] [PubMed]
29. Li, Q.; Zhang, Q.; Huang, H.; Zhang, W.; Chen, W.; Wang, H. Secure, efficient, and weighted access control for cloud-assisted industrial IoT. *IEEE Internet Things J.* **2022**, *9*, 16917–16927. [CrossRef]
30. Novo, O. Blockchain meets IoT: An architecture for scalable access management in IoT. *IEEE Internet Things J.* **2018**, *5*, 1184–1195. [CrossRef]
31. Zhang, Y.; Kasahara, S.; Shen, Y.; Jiang, X.; Wan, J. Smart contract-based access control for the internet of things. *IEEE Internet Things J.* **2018**, *6*, 1594–1605. [CrossRef]
32. Ding, S.; Cao, J.; Li, C.; Fan, K.; Li, H. A novel attribute-based access control scheme using blockchain for IoT. *IEEE Access* **2019**, *7*, 38431–38441. [CrossRef]
33. Liu, H.; Han, D.; Li, D. Fabric-IoT: A blockchain-based access control system in IoT. *IEEE Access* **2020**, *8*, 18207–18218. [CrossRef]
34. Rouhani, S.; Deters, R. Blockchain based access control systems: State of the art and challenges. In Proceedings of the 2019 IEEE/WIC/ACM International Conference on Web Intelligence, Thessaloniki, Greece, 14–17 October 2019; pp. 423–428.
35. Stock, F.; Kurt Peker, Y.; Perez, A.J.; Hearst, J. Physical visitor access control and authentication using blockchain, smart contracts and internet of things. *Cryptography* **2022**, *6*, 65. [CrossRef]
36. Li, Z.; Li, J.; Zhao, S.; Chen, X.; Feng, K.; Wang, W. A blockchain-based lightweight identity authentication scheme for the IEDs of security and stability control system. *PLoS ONE* **2022**, *17*, e0265937. [CrossRef]
37. Abdi, A.I.; Eassa, F.E.; Jambi, K.; Almarhabi, K.; Khemakhem, M.; Basuhail, A.; Yamin, M. Hierarchical blockchain-based multi-chaincode access control for securing IoT systems. *Electronics* **2022**, *11*, 711. [CrossRef]
38. Hu, L.; Huang, Z.; Deng, F.; Yan, K.; Liu, J. Towards a Location Aware Semantic Access Control Approach for Mobile Computing. In Proceedings of the 2016 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI), Beijing, China, 20–21 October 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 485–490.
39. Hong, J.; Xue, K.; Xue, Y.; Chen, W.; Wei, D.S.; Yu, N.; Hong, P. TAFC: Time and attribute factors combined access control for time-sensitive data in public cloud. *IEEE Trans. Serv. Comput.* **2017**, *13*, 158–171. [CrossRef]
40. Balani, N.; Ruj, S. Temporal access control with user revocation for cloud data. In Proceedings of the 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, Beijing, China, 24–26 September 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 336–343.
41. Ameer, S.; Benson, J.; Sandhu, R. The EGRBAC model for smart home IoT. In Proceedings of the 2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI), Las Vegas, NV, USA, 11–13 August 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 457–462.

42. Zhang, J.; Wang, Z.; Zhao, P.; Gao, M.; Sun, S. Comparative Attribute Access Control Scheme Based on Spatio-temporal Constraints in Cloud. *Int. J. Netw. Secur.* **2022**, *24*, 469–481.
43. Booch, G.; Rumbaugh, J.E.; Jacobson, I. *The Unified Modeling Language User Guide—Covers UML 2.0*, 2nd ed.; Addison Wesley Object Technology Series; Addison-Wesley: Boston, MA, USA, 2005.
44. Monge, A.; Jewett, T. *Database Design with UML and SQL*, 4th ed.; California State University: Long Beach, CA, USA, 2006.
45. Freeman, A. *Pro ASP.NET MVC 5 Platform*; Apress: Berkeley, CA, USA, 2014.