Final Year Project

A Tool for Engineering Adaptive Authentication

Student Yifu Yang

Student ID: 17204587

A thesis submitted in part fulfilment of the degree of

BSc. (Hons.) in Computer Science

Supervisor: Liliana Pasquale, Alzubair Hassan

UCD School of Computer Science

University College Dublin

April 29, 2022

Table of Contents

Details

GitHub Repositories:

1. The server-side of my final year project: https://github.com/YifuYANG/apis_for_FYP
2. The client-side of my final year project: https://github.com/YifuYANG/MyFYP

Abstract

Authentication is a mechanism for confirming a user's identity. Authenticated identities are those who can show a valid credential. In this report, we introduce an adaptive authentication system that is frequently modified in reaction to changes in the security risk or the behavior of the user. We mainly focused on the development of a tool implementation that supports the most appropriate authentication method depending on the execution context. Our approach takes into account the simulation of three scenarios, for each scenario we implemented a corresponding authentication mechanism.

Chapter 1: Project Specification

The fourth industrial revolution is bringing changes to our society and economy. People would send greetings online rather than write letters; prefer online shopping to traditional one, and spend more time on the internet. While enjoying much convenience of interconnectivity, we are facing lots of challenges in both real and virtual worlds, such as privacy disclosure, and cybercrimes. As a result, the authentication mechanism is becoming increasingly crucial.

Authentication systems identify and enforce appropriate ways for determining if someone (user) or something (device) is authorized to utilize a service or resource[1]. Unfortunately, with the revolution of technology, traditional authentication system such like singly used username and password validation is no longer enough to support the domains of users at this stage. Liliana et al. explores the idea of Adaptive authentication systems in their research. A system that adapts the authentication method in response to changes in the security risk or the user's behaviour[1].

This project aims to create a software prototype to demonstrate an adaptive authentication system based on the changing of security risks and contextual factors. However, choosing different authentication methods based on contextual factors in different domains can be challenging since the priorities of requirements can be altered.

Therefore, the current challenges for adaptive authentication tasks can be addressed as follow:

1. Identify the contextual factors that can affect the feasibility of authentication methods. (location, time, speed, etc)
2. Identify how different authentication methods can affect the satisfaction of the requirements. (performance, confidentiality, and both performance and confidentiality)

The related work of this report can be classified into four categories in terms of adaptive authentication system: Introduction of UAP and Trust engine, combination of parameterization and context awareness, risk-based adaptive authentication and Continuity authentication, the most related UAP and Trust engine works are 3.1 Analysis of User Behavior, 3.2 implementing random canvas fingerprinting as user attributes factor and 3.3 Adaptive Authentication to determine login attempt penalty from multiple input sources. 3.4 Context Sensitive Adaptive Authentication came up with a new concept called the combination of parameterization and context awareness, this concept overcomes the defects of traditional security system. In 3.6 Risk-Based Adaptive Authentication for Internet of Things in Smart Home eHealth authors explained a concept similar to that of a trust engine. The concept of Continuity authentication was proposed at 3.7 Improving the Security and QoE in Mobile Devices through an Intelligent and Adaptive Continuous Authentication System and 3.8 Progressive authentication: deciding when to authenticate on mobile phones.

Extended works research: 3.5 TreasurePhone: Context-Sensitive User Data Protection on Mobile Phones. this research introduced the notion of spheres, A sphere symbolizes the user's privacy requirements for data on her mobile phone in a specific environment.

There is one key publication on which we primarily concentrate and strive to improve:

Engineering Adaptive Authentication[1] - The authors of this article proposed a framework to characterize the adaptive authentication problem and facilitate the implementation of adaptive authentication systems, based on past research.

Main activities

The prototype will monitor the contextual factors and the requirements of different scenes to change the authentication methods. The prototype will support two main activities:

1. *Monitoring*: Evaluate the contextual factors such as user's location, type of device, network, and other factors in order to calculate the risks that the user is exposed to before connecting to the system. For example, if a user's customary address is Dublin, and wants to login from a different location or device, the system should enforce the use of a different and stronger authentication method.
2. *Execution*: Identify and enforce an effective authentication that maximizes the satisfaction of relevant requirements, such as security, usability, and performance, and reduces security risks. The software prototype will be demonstrated using an Internet of Vehicles example.

Chapter 2: **Introduction**

Authentication is one of the most important aspects of information system security. It verifies a user's credentials to check if the user is qualified to use certain services or resources [1]. Authenticated identities are those who can show a valid credential. Users are then awarded permissions depending on their verified identities. The Authentication system has undergone multiple updates in recent years and has been fine-tuned to perfection. Usually, the single-factor verification method or the two-factor verification method is used for identity Authentication. The single-factor authentication depends on three types of credentials: what you know, what you have and what you are[2], for example, uses a user name/password for authentication. Two-factor authentication is usually performed by SMS verification and fingerprint verification which depends on what you have and what you are[2]. Regardless of whether it is single-factor authentication or two-factor authentication, the user is constrained to the specified security verification strategy and cannot adjust the tailored security verification strategy to meet the user's specific needs.

Adaptive authentication, commonly known as risk-based authentication, is a technique that attempts to adapt the authentication methods to match the domains of the current scenario based on contextual factors. Its purpose is to reduce risk while implementing strong authentication where it is most needed. For example, user who is trying to access public information should use weak type authentication, but for accessing private data such as bank detail or email should require strong authentication, since the data is crucial.

It is difficult to choose an invariant and efficient authentication method since the demand cannot be predicted as the scenario changes [1, 3] for example, An ambulance suddenly needs to overtake while obtaining road information, the system's authentication criteria and contextual factors must both be modified simultaneously. Previous work on adaptive authentication, provides only rudimentary information on how to build adaptive authentication systems in a systematic manner [1, 4, 5]. As a result, a number of challenges were discussed by Hassan et al. [1] as follow:

1. **how contextual factors can affect the feasibility of authentication methods**: changes in contextual factors can change the priorities of authentication requirements
2. **how different authentication techniques affect requirement satisfaction**: certain authentication requirements may not be feasible in certain situations

3. **how to monitor all contextual factors changes continuously to detect specific situations**: continuously identify the contextual factors that are able to affect authentication priorities

The aim of this project is to adjust authentication methods based on the changing contextual factors (e.g, speed, time, traffic) and requirements (confidentiality, performance, both confidentiality and usability). The objectives of our research are the determination of the authentication methods, identifying contextual factors that can affect authentication priorities based on the scenario requirements, and combining contextual factors and authentication methods to achieve authentication system adaption. To motivate the adaptive authentication challenge we implemented and explored three adaptive security scenarios using an Internet of vehicles case study based on Hassan et al' idea[1].

The structure of the project is as follows: Charter 3 introduces relevant efforts that aided us in grasping the fundamental concept of adaptive authentication. Charter 4 outlines our project's timeline. Charter 5 discusses the high-level overview of the architecture for each approach and gives viewers a full perspective of our solutions. Charter 6 explain how we implement our project with more specific details including code samples and sequence diagrams. Chapter 7 evaluates the performance of our application. Chapter 8 summarizes the shortcomings of our project and offers recommendations for further development.

Chapter 3: Related Work and Ideas

Adaptive authentication creates a profile for each user that includes information like the user's geographical location, registered devices, role, and more. Every time someone tries to authenticate, authentication methods will alter based on the domains of the current scenario and contextual factors. The user may be forced to submit extra credentials or allowed to use fewer credentials depending on the requirements. The milestone can be broken down into two stages: monitoring and execution. At the monitoring stage, we will assess the risk level based on contextual factors and requirements so that the system can determine whether the current user requires additional identification. At the execution stage, we will determine the best authentication methods that will best suit the system.

3.1 Adaptive Authentication based on Analysis of User Behavior

Unified authentication simplifies the login process by allowing users to gain full access control on manager instances with a single set of login credentials. Based on the knowledge of unified authentication Bakar and Haron [2] introduce the Unified Authentication Platform (UAP) authentication system by providing the concept of a trust engine, a trust engine changes the authentication technique based on a user's physical or behavioural characteristics, security risk and confidence level(Current user credibility base on the risk level), so as to consider the attributes (such as the user's device) from the user's typical profiles, which have been previously analyzed and saved while determining authentication decisions.

Bakar et al.[2] provided two basic processes that focus on the system: Pattern Generating and Trust Evaluating that focus on two main factors which are physical and behavioural characteristics of a user, in order to make adapted methods. (1)The pattern generating process is responsible for analyzing users' behavior from previous records such as IP addresses and producing the corresponding user attributes profile that will be used in the next trust evaluating process called pattern generating process. This process analyzes past records for all users in the system, which means it consumes a lot of computing power. As a result, the pattern generating process is only scheduled to run at midnight. (2)The trust evaluating process is the second step in adaptive authentication; it analyzes, decides, and acts on each user login request. The trust calculator is an important aspect of the evaluating process. Which will calculate the user's total trust score by comparing the current contexts processed by the context collector with the user attribute profiles retrieved from the pattern storage.

3.2 Adaptive Authentication: Implementing random canvas fingerprinting as user attributes factor

Canvas fingerprinting technology is used by websites to track visitors who view their content. In[6] the authors introduced and discuss the reason why a random canvas fingerprint can be unique so that we can use canvas fingerprint as one of the attribute elements that UAP will evaluate when analyzing user-profiles and incorporating them into the evaluation process before providing access to the user. Canvas fingerprinting is a user tracking technique that takes advantage of HTML5's drawing feature, which is analyzed by a Trust Engine while determining the user's trust level. The purpose of the trust engine is to enable adaptive control based on security risk and assurance level.

However, Daud et al.[6] proposed that the canvas fingerprinting value is not unique for each user using the same operating system and hardware specifications. This means that if a user uses the same credential on a different workstation, the Trust Engine will flag it as anomalous behavior and prompt the user to give an alternative authentication method. But, since the canvas fingerprint value is the same for every user with the same operating system and machine specifications, the system will not ask the user to provide an extra authentication mechanism in this case. In addition, if hackers know the credential for a certain user, they may be able to log in to the system using a different workstation. To overcome this constraint, in this paper the authors allow the system to choose a random fingerprinting value that users will use, reducing the likelihood that the system will use the same fingerprint for each user.

Unique canvas fingerprints gave a good example of contextual factors and what a unique factor should look like[6]. Instead, in this project, a different set of contextual factors such as network, location, time of the day, lighting, user activity will be considered.

3.3 Adaptive Authentication to determine login attempt penalty from multiple input sources

Mi-UAP is designed to manage front-end application authentication using an established protocol, Secure Assertion Markup Language (SAML). This paper[7] will explain the way that overcomes the authentication from any suspicious platform environment and introduce an authentication

platform: Mi-UAP. In the study by Daud et al.[7] there are 4 main components in Mi-UAP. Which are UAP Gateway, Web Application Server, UAP Server, and Trust Engine.

We have been discovering the Trust Engine in the past two papers[2][6], explained what should the factors look like and how the engine would take those factors and put them into the evaluation step, in this paper we will shift our focus on how trust engine uses user's behavioural characteristic as factors to make reaction to users.

The aim of the trust engine was introduced by Daud et al.[7], it is to support adaptive control based on security risk and assurance level. The Engine analyzes user behavior and calculates attribute factors such as geographical location and browser type before deciding whether to provide the user access to the system or require them to submit additional authentication methods.

However, the current implementation, on the other hand, Daud et al.[7] bring forward a drawback. Since we're working with a small database and the system's speed is based on how much data we have, the more data we have, the more filtering methods we may use. As a result, we should consider integrating the solution with more IDS technologies in the future.

The research by Daud et al focuses on the different reactions of the system depending on different factors and provided the logic of how to detect suspicious behaviours from users through those factors. In order to make the system adaptive, using similar logic to set up an evaluation process should be considered.

3.4 Context Sensitive Adaptive Authentication

Traditional security systems are unable to be flexibly changed to new limitations.This limitation stems from two major flaws: 1)the inability to implement parameterization and 2)the inability to take context information into account. To overcome this defect Hulsebosch et al.[8] combines parameterization and context-awareness to control security adaptation, by the meaning of combination is that we can optimize the security functionality for a system that is able to maintain the desired security level and respond to new security constraints that may arise from changes in the situational context by constantly monitoring and analyzing context information.

To achieve the idea we need to focus on such factors that connect context like location, Hulsebosch et al.[8] gave an example a system can use location information to determine and dynamically adapt the authentication level of a user. The chance of the user being in a given location of interest l, which is expected to be the location from which the user forwards his access request, is calculated using location data from many different location sensors. The result is used to determine the user's authentication level and, if necessary, adjust his authorization level. We'll need to use a framework called Context Management Framework to get sensor location data (CMF).

The CMF enables the processing and exchange of heterogeneous context information collected from various sensors, is distributed over multiple administrative domains, and stems from different protocol layers. Hulsebosch et al.[8] provide examples of context information supplied by the CMF including location coordinates via GPS receivers, WLAN access points associations, RFID reader data, BT scan measurements, desktop keyboard typing, and Outlook Calendar meetings.

This study focuses on the use of parameterization and context awareness in conjunction with security adaptation to maximize a system's security functionality. however, we will only focus on the factors that are able to connect context and also are easier for a mobile app to monitor with such as user's location, instead of those factors mentioned in this paper (user habits, mental state, social environment, or task-related activities).

3.5 TreasurePhone: Context-Sensitive User Data Protection on Mobile Phones

Mobile phones have been around the world for over a decade, the concept of Treasurephone was first introduced by Seifert et al[9]. They present the Treasurephone that uses the phone's location or, on the other hand, the user's location as the primary factor to assist sensitive data protection by allowing the user to designate so-called circles. TreasurePhone uses locations for automatic sphere activation and supports interaction with the user's environment to activate appropriate spheres on the go, reducing the risk of unintentionally disclosing sensitive information because it allows users to secure their data in each context in a sophisticated way using their mobile phone.

TreasurePhone's concept is built on users' desire to secure and manage the privacy of their personal information kept on their mobile phones. Seifert et al.[9] propose the notion of spheres, which is based on Goffman's faces (a face defines what information a person shows to a certain audience). A sphere symbolizes the user's privacy requirements for data on her mobile phone in a specific environment. That is, the user can specify which apps, such as e-mail clients or photo viewers, are available in a given sphere, as well as what data is and is not accessible.

In order to achieve this idea, There are three types of relative work proposed by Seifert et al. [9]that can be categorized: 1) conceptual work on data privacy for mobile devices, 2) authentication mechanisms for mobile phones, and 3) context-dependent adaptive mobile devices; we will only discuss the first and third categories in this paper. 1) Stajano came up with a solution to the privacy difficulties that come from sharing (intentionally or unintentionally) a personal digital assistant (PDA). He describes a system for PDAs that is based on the observation that some data and programs can be utilized by anybody who has access to the PDA. However, other applications and data should only be accessible by the device's rightful owner. Authentication is required to access these private regions or "hats," which protects the user's privacy. 2) Siewiorek et al. offer SenSay, a mobile phone that modifies its behavior depending on the circumstances. Based on the results, this system analyses data collected by multiple sensors and identifies the user's present context.

3.6 Risk-Based Adaptive Authentication for Internet of Things in Smart Home eHealth

Gebrie and Abie[10] propose a novel risk-based adaptive authentication in this paper. it is the method for WBAN in Smart Home eHealth environment with a similar idea of Trust Engine component that we discuss in previous papers(Adaptive Authentication based on Analysis of User Behavior)[2]. The method entails continuously monitoring and analyzing the behaviors of the user and devices and then picking authentication or re-authentication protocols based on the security risk. It also analyzes the resource requirements of the selected authentication protocols with the available resources of the authenticating device to determine whether or not the authentication process should be offloaded.

User authentication is an essential component of any security infrastructure. Users can be authenticated in a variety of methods, including utilizing something the user knows, something the user owns, something the user is, something the user does, where the user is, and any combination of these[10] as figure 3.1 below.

Authentication	Types
Something you know	Password, PIN, Personal number, Phone number, date of birth, etc.
Something you have	Tokens, Smartcards, Bank Card, Passport, Driving license, etc
Something you are	Biometrics: Physiological biometrics such as fingerprint, facial recognition, iris-scan, hand geometry, retina scan, etc., and Behavioural biometrics such as voice recognition, keystroke scan, signature-scan, gaits, et
Something you do	User behaviours patterns, bank transactions, travelling, calls, social media logs, etc.
Combinations	Any combinations of the above (aka multifactor authentication, e.g. PIN-enabled bank card)

Figure 3.1: Authentication Types.[10]

Gebrie and Abie[10] came out with a variety of solutions available for each of these authentication types as the last factor suggests, ranging from single factor (e.g. user name and password) to multi-factor authentication (using different types of authentication mechanisms). The goal of this paper is to adapt authentication techniques dynamically in response to changes in the environment.

To increase the flexibility and security of authentication by dynamically adjusting the authentication technique in response to context changes. Two Authentication mechanisms have been introduced, which are Adaptive Authentication and Risk-based Authentication[10]. we will focus on Risk-based Authentication since Adaptive Authentication can be referred by Risk-based Authentication. Risk-based authentication calculates the risk score associated with the user's current action using contextual and previous data. The risk score is determined in real-time using a set of rules that can be applied to authentication choices. The general purpose of risk-based authentication is to acquire information from the user's surroundings, compare it to a known user profile, and determine whether the user requires extra authentication[10].

A new idea of risk score was discussed in this section, which we should also consider having a similar mechanism in our own implementation. the mechanism should be able to monitor the context changes in order to increase the flexibility of authentication and level of security.

3.7 Improving the Security and QoE in Mobile Devices through an Intelligent and Adaptive Continuous Authentication System

This section[11] gives us a better understanding of the lifecycle of continuous authentication systems, it aims to find the way to identify the owner of a device permanently by using continuous authentication systems. The system considers data from application usage statistics and sensors to predict users' behaviors as factors.

When compared to older approaches, the fact that the user is continuously authenticated rather than periodically authenticated contributes to a higher level of security and confidence. This strategy has various advantages, such as reducing the need for credentials during authentication

operations using continuous authentication systems.

This paper describes in detail the four phases which are Feature engineering, Acquisition of behavioural data and dataset generation, Computation of the authentication level, and Automatic adaptability to new behaviours making up the design of adaptive continuous authentication system[11] as figure 2 below.

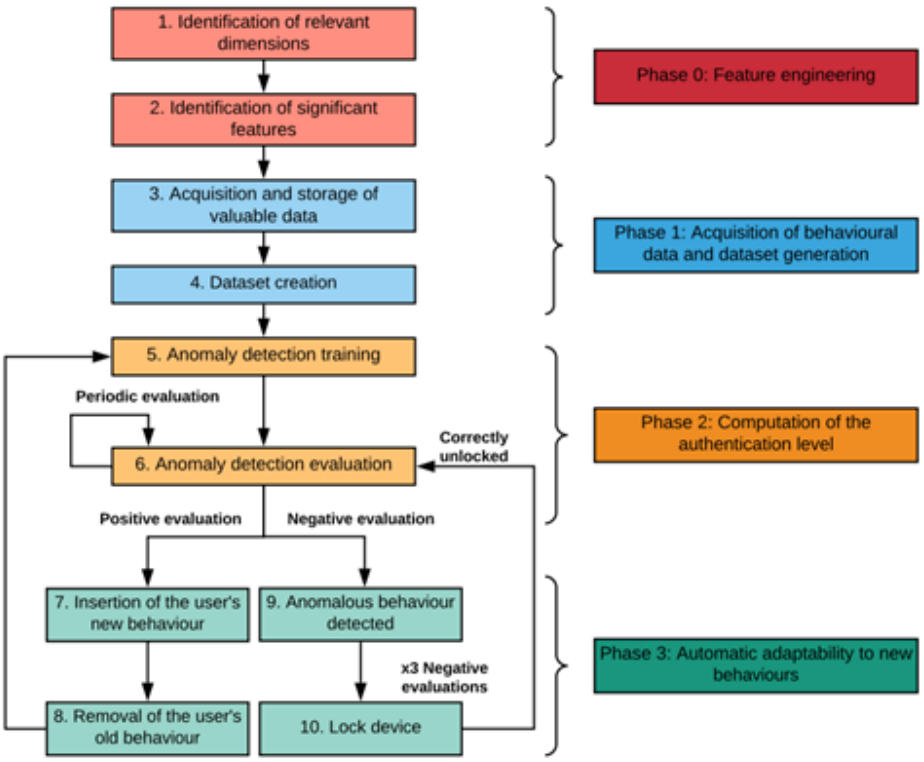

Figure 3.2: Phases and processes of the proposed adaptive and continuous authentication system.[11]

In trying to find a better solution of adaptive authentication, having the user permanently authenticated is becoming increasingly popular. However, for a mobile application, it is impossible to have such permanent credentials. In response to this, we will have a semi-permanent authentication to reduce the process so users do not need to continually provide credentials during the authentication processes, and this function can be turned off by the owner.

3.8 Progressive authentication: deciding when to authenticate on mobile phones

Riva et al.[12] introduced a new authentication model for mobile phones called progressive authentication. In their research, they recommend mobile systems that gradually authenticate users by collecting credentials about them on a regular basis, ensuring that the user is authenticated with a high level of trust at any time they try to log in to the system. This strategy will reduce the frequency of such events, resulting in a significant reduction in the user's authentication overhead.

the system is designed for phones and mobile devices with rich sensing capabilities, the system consists of two levels as figure 3 below[12], which are monitoring levels and processing levels similar to the Trust Engine.

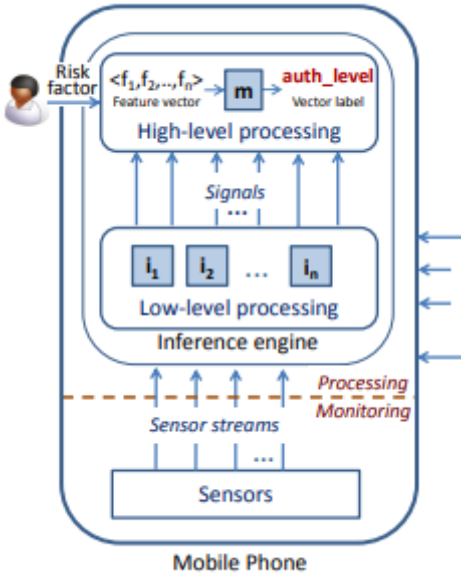

Figure 3.3: Progressive authentication architecture.[12]

Riva et al.[12] proposed that at the monitoring step, the system gathers sensor factors, such as the user's biometric , and passes it into the processing stage, which determines if the user requires additional authentication or whether the authentication should be canceled.

Due to the shortage of convenience, user-defined settings provided Riva et al's with an idea to have a system that exposes risk factors to users so that users can adjust how aggressively they want to trade security for convenience by themselves. However, the higher the risk factor, the higher the convenience, but also the higher the security risk.

Chapter 4: Project Workplan

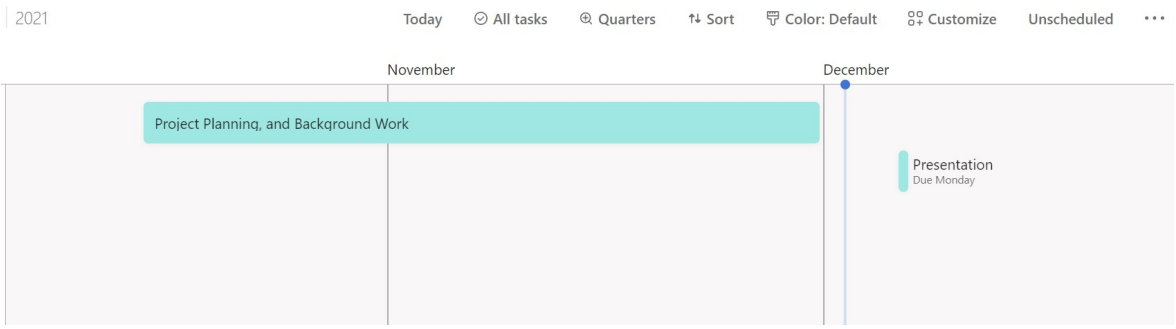

Figure 4.1: work plan.

The project's primary focus will be project planning, background work, and presentation, as these tasks form the project's basis. Studies will be undertaken in order to fulfill future assignments with fewer issues.

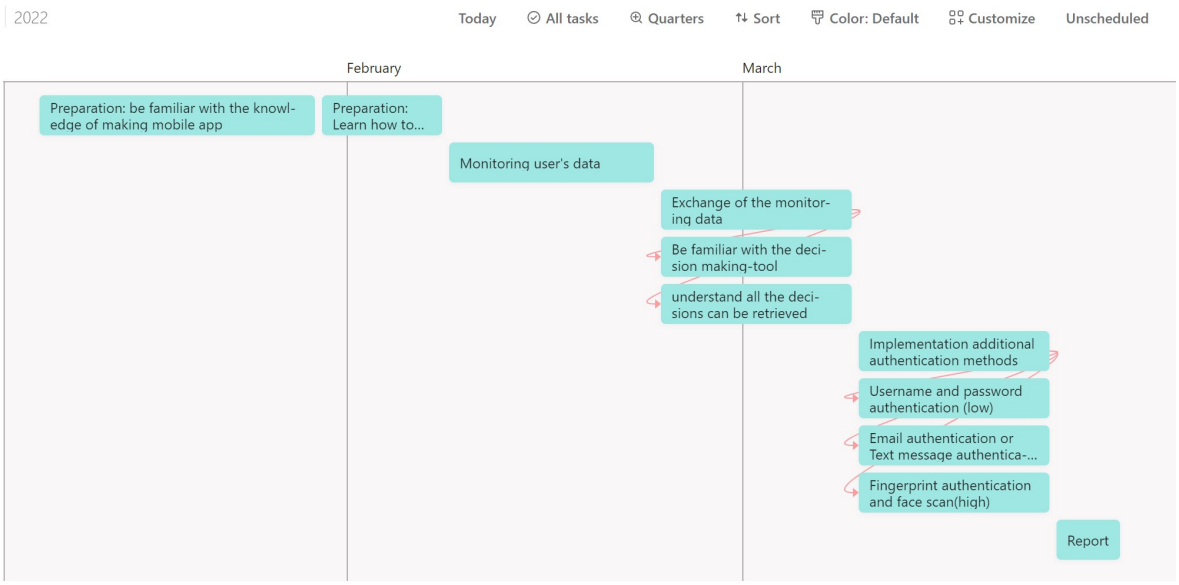

Figure 4.2: work plan.

The remaining tasks are preparation, monitoring of contextual factors, implementation of additional authentication methods based on the requirements and contextual factors, and a final report. Each task will take two to three weeks to complete. It begins with preparing the technique for developing an Android app, followed by acquiring user data such as location or device type that the user often uses to log into the system. Finally, we use the contextual factors we observed to identify and enforce an effective authentication that maximizes the satisfaction of the requirement in each scenario.

Chapter 5: Project Approach/Design

This section discusses the high-level overviews of the architecture for each approach and gives viewers a full perspective of our solutions. We will also discuss the unsolvable problems we ran across while deciding on contextual factors and authentication techniques, as well as some further advice for developers.

Our approach of this project is based on Hassan et al' idea[1]: a simulation of an ambulance authentication system, the simulation can be divided into three scenarios according to different requirements:

- 5.1 The ambulance is trying to acquire road traffic information: The ambulance needs to obtain road traffic information in order to arrive at the hospital as quickly as feasible. To accomplish this, it uses Vehicle-to-Roadside-Units topology to connect with the nearest roadside units(see Figure 5.1a).
- 5.2 The ambulance is trying to overtake another vehicle: The ambulance is attempting to pass the red car (see Figure 5.1b). To accomplish this, the ambulance needs to use Vehicle-to-Vehicle communication architecture to extract the information about nearby cars' respective locations (red and blue cars).
- 5.3 The driver is trying to acquire patient information: The ambulance driver is accessing patient's information at a junction using Vehicle-to-Infrastructure of cellular networks (see Figure 5.1c)

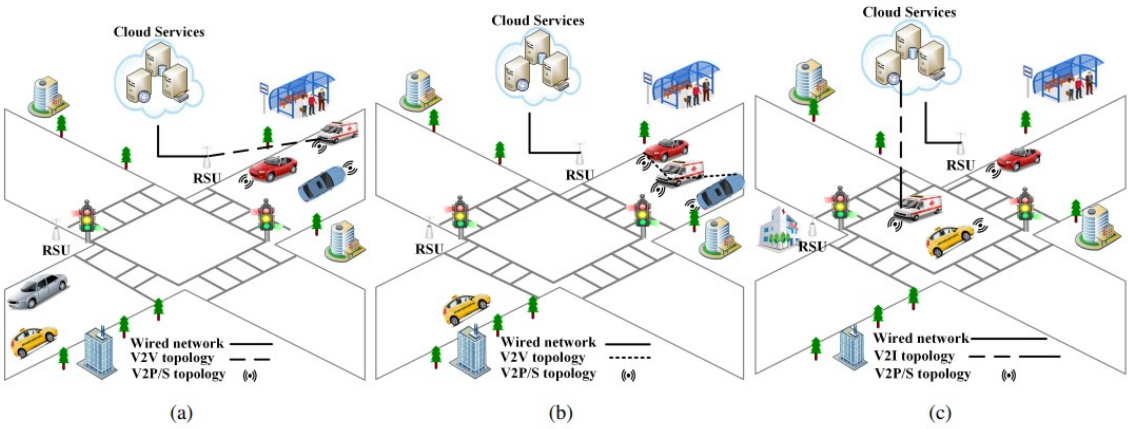

Figure 5.1: Adaptive Authentication Scenarios[1]

For each of the scenarios, we build an Android activity and a corresponding web application:

- **Android activity**: represents the client-side of the project to monitor the contextual factor and enact an appropriate authentication
- **Corresponding web application**: containing several Restful API represent the server-side of the project to verify user's identity remotely.

5.1 Scenario 1: The ambulance is trying to acquire road traffic information

When trying to acquire road traffic information to reach the destination, the ambulance needs to communicate with the nearest roadside units using a Vehicle roadside units topology. In this case, confidentiality has higher priority over performance since traffic information could be highly sensitive, no contextual factors should affect the authentication methods because of the high confidentiality requirement.

In order to satisfy the high level of security requirement and ensure the driver is not distracted, an SSL authentication (certificate-based authentication) will be applied to this scenario.

SSL authentication encrypts and hides transmitted data, ensures that data is not altered in transit, there are two type of SSL authentication available:

- One-way authentication (One Way SSL): only one party verifies whether the other party is legal, usually the client verifies the server.
- Both-way authentication (Both Way SSL): mutual detection is required between the client and the service segment.

We determined that the ambulance should be the center of this authentication, the server should verify that whoever every trying to access the server is a registered ambulance. Obviously, the One Way SSL is inapplicable because it contradicts our viewpoint.

To satisfy our decision at this point, the both way SSL authentication will be used. The process of both way SSL authentication at displayed in Figure 5.2 below.

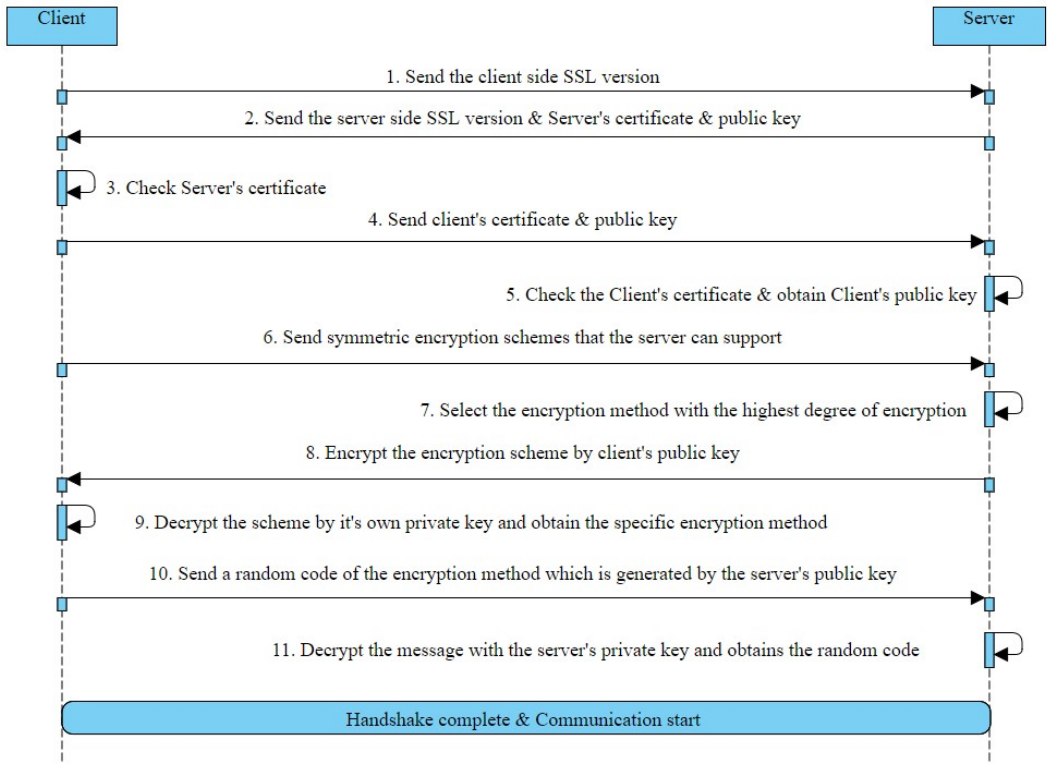

Figure 5.2: Both-way SSL authentication process

To generate certificates. Our first thought was the digital certificate production tool built into the JDK: keytool. keytool can generate self-signed digital certificates. The so-called self-signed means that the certificate can only guarantee that it is complete and has not been illegally modified[13]. But there is no guarantee to who this certificate belongs, and this verification has a drawback: a confirmed copy of the certificate must be retained on each server in order for them to be connected. All clients will need to reinstall these copies after the server changes the certificate. For larger applications, this is not acceptable. Therefore, the certificate chain is required for both-way authentication.

However, services using CA-certified certificate chains are not free. Therefore, it is necessary for us to act as a root account of our own certificate authority and sign the CA certificate to publish the key pair. Then when deploying the application, we merely need to deploy the private key of our self-published CA certificate on all nodes to complete the verification. For self-published CA certificate generation, OpenSSL is required.

We will deploy the certificates to the corresponding server and client once they are ready. As shown in Figure 5.3, there are four components that make encrypted the connection possible, key store and key manager store their respective certificates (e.g, the server-side key store stores the server's certificate) which decide what certificate to send to the other party while trust store and trust manager store their corresponding certificates (e.g, server-side trust store stores client's certificate) which determine whether the certificate from the other party is credible.

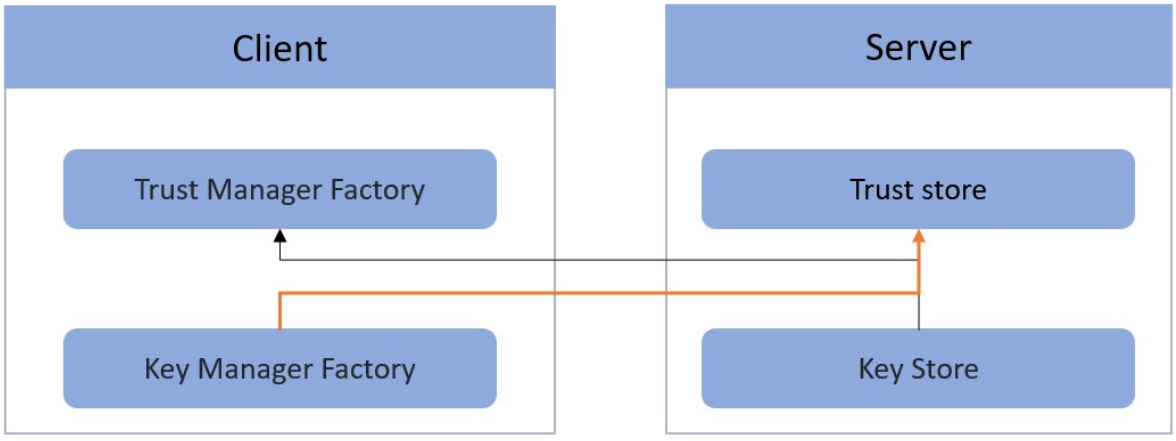

Figure 5.3: Main components for SSL authentication

To represent if a client connects to the server-side system successfully, we designed one simple restful API which will return an accepted HTTP status back to the client, once response code 202 is returned to the client, it indicates that the connection is established and the client can communicate with roadside units safely.

5.2 Scenario 2:The ambulance is trying to overtake another vehicle

When the ambulance is trying to overtake another vehicle, it needs to exchange its and the relevant vehicle's coordinates with the server to obtain the relative distance. performance requirements will be the first priority for safety reasons at this point. In order to maximize the efficiency, we will determine the type of authentications based on contextual considerations.

The contextual factors that can influence authentication method in this simulation are as follows:

- Location
- Time
- Speed
- Traffic
- Device ID

According to the notion of drivers should not be interrupted while driving, the action of driving is best represented by The contextual factor: speed. As a result, it can be concluded that the speed and the traffic factors are the best option for this scenario. However, since Google does not provide any services that allow developers to directly access traffic data, we are unable to demonstrate how this context aspect affects the authentication process in this project. Our next step will be selecting appropriate authentication methods, same as the previous discussion the authentication methods should not interrupt the driver or keep disruptions to a minimum and should also minimize the time to be performed.

Possible authentication methods in this simulation are as follows:

- Username and Password base authentication
- License comparison based authentication
- Biometric based authentication
- Certificates based authentication

Username and password based authentication was excluded first because this authentication method requires frequent user interaction, which is not in line with the concept of not disturbing users. Certificate-based authentication is similarly ineffective in this situation since verifying the identity of the vehicles on a remote server can take excessive time. The remaining methods are Biometric based authentication and License comparison based authentication which is going to be used in this scenario.

To satisfy the performance to the greatest extent, we need to evaluate the authentication method that should be executed according to the speed factor. Our system will determine if the ambulance is moving, and then decide what action to take. License comparison based authentication should be selected if the ambulance is in motion, as this method does not require driver interaction and has the quickest execution time. On the contrary, biometric authentication should be selected, this solution involves succinct user intervention and fits the requirements of this scenario in terms of execution time.

We only wish qualified vehicles to use our services, so after the authentication process has been completed, the system should attempt to pass the server-side verification by sending the previously-stored license plate number and the corresponding password to the server and then obtain a unique token. Finally use the token to retrieve the distance information. The main components which construct the operation are shown in Figure 5.4 below:

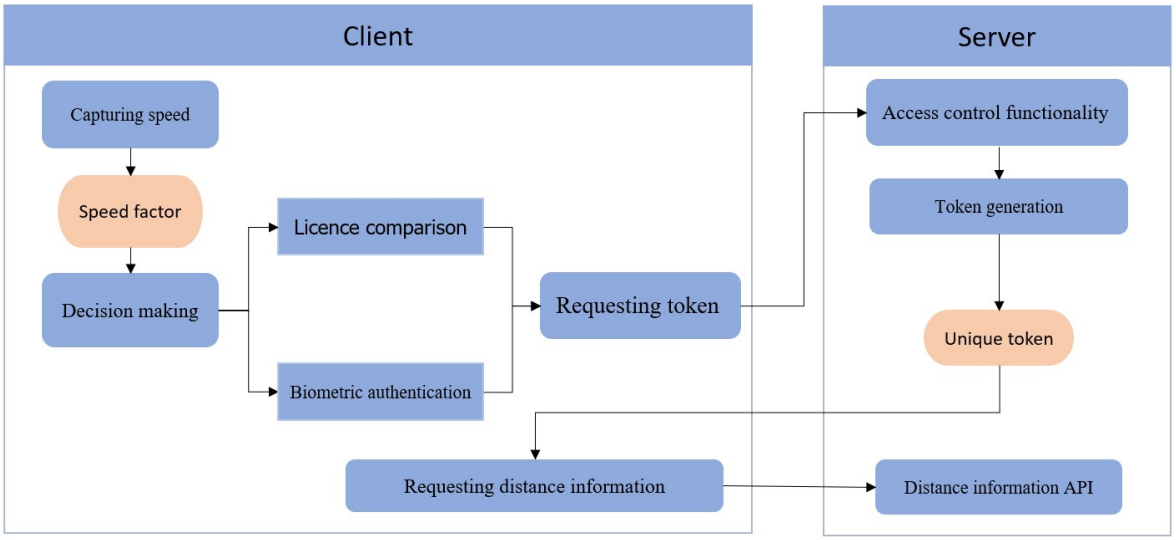

Figure 5.4: Main components of the operation

In our original approach, we attempted to let the server handle the decision making stage then simulated a third-party organization to support the license comparison authentication, we rapidly discovered that such a structure was incompatible with the scenario's central notion as it requires multiple round-trip exchanges of results (contrary to the idea of minimizing the time to perform authentication). As a solution, we delegate the responsibility of license authentication and decision-making to the client.

Further consideration

For further development for this scenario, the developers should contact the Google team to acquire real-time traffic information so that the system may combine the traffic factor with the speed factor to achieve self-adaption more accurately.

5.3 Scenario 3: The driver is trying to acquire patient information

Maintaining high usability and confidentiality criteria are equally critical when drivers attempt to obtain patient information on their way to the hospital because they are accessing sensitive information while need to arrive at the hospital as soon as possible. Two-factor authentication will be utilized in this scenario to balance those requirements.

To reduce authentication time while maintaining high security standards we had to surrender some of our previous beliefs. As we have previously discussed at 5.2, the authentication method we selected should not cause too much disruption to the driver while driving. However, since patient information can be highly sensitive, we will use a combination of username and password based authentication and biometric based authentication to satisfy this scenario.

To cut driver contact time, we will leverage contextual factors to govern biometric authentication methods, we observed that the speed factor we previously selected does not apply to this situation

so we will use the time factor instead. Our system should calculate the time to determine whether it is day or night, and force users to use different biometric authentication methods at different times.

- Day time: face scan(minimizes user interaction)
- Night time: fingerprint(face scans may be ineffective due to lighting concerns)

The main components of two-factor authentication can be found in Figure 5.5 below:

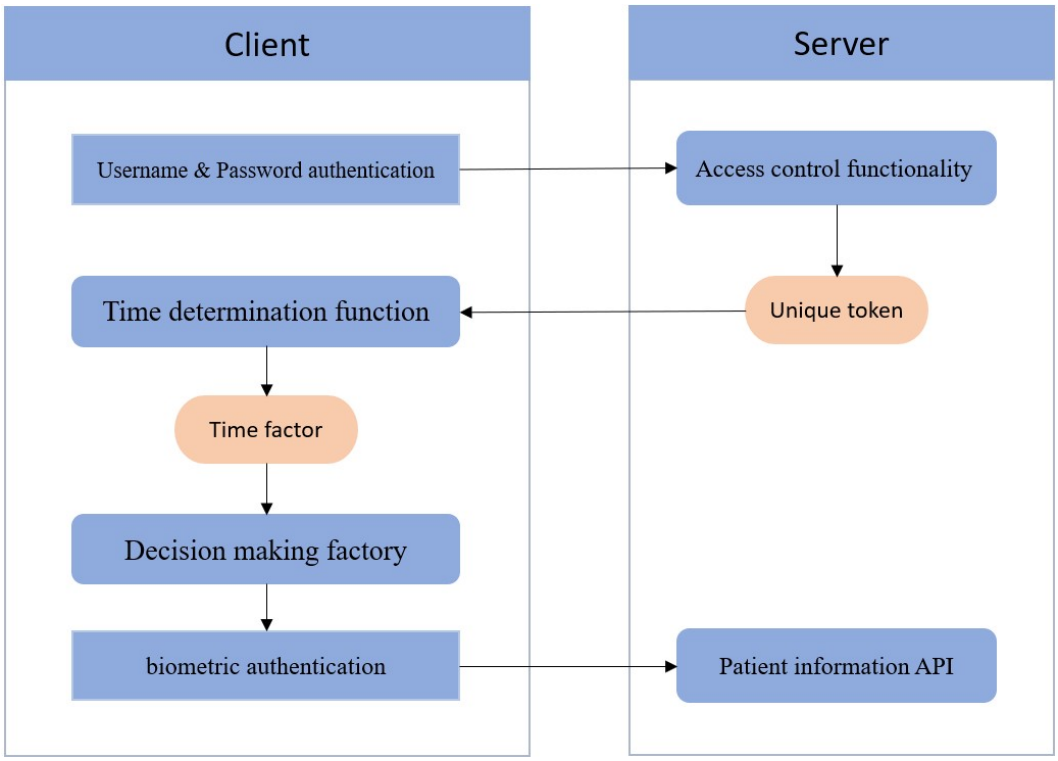

Figure 5.5: Two-factor based authentication

However, during the early preparation, we noticed that we are unable to specify which biometric authentication approach will be used since the Android Biometric Prompt does not provide the developer with the ability to select a specific biometric authentication method (the developers are asked to use the terminology "Biometric" instead of "Fingerprint" or "Face"), their system considers fingerprint authentication to be more reliable than facial scanning by default, which means fingerprint recognition will be used for biometric authentication in the vast majority of circumstances.

Several studies have demonstrated that the developers may force the system to reduce the strength of the biometric authentication methods by setting authorized authenticators to weak types, but this varies depending on the device hardware and system version. As a result, we were unable to make the system adaptive for this case based on the time factor since we were unable to evaluate the accuracy of our assumption for the following reasons:

1. face unlock is only available on the Pixel 4
2. Android emulator only supports fingerprint verification while we don't have access to an actual Pixel 4 device

Further consideration

For further development for this scenario, the developers should consider setting authorized authenticators to weak types in order to force the system to reduce the strength of the biometric authentication methods, then test the accuracy of the result before moving on to the next stage. by doing so the users can enable face recognition and the system can be adaptive based on the time factor (fingerprint during the nighttime, face scan during the daytime).

Chapter 6: Implementation

In this section we will describe the realization of the ideas and concepts described in previous sections, but with more specific details. The section will include code samples, operation sequence diagrams, and detailed descriptions of authentication methods, we also discuss the issues we encountered during the implementation and provide a corresponding solution.

Our original intention was to design an Android Auto application and then test it using an emulator as it is an in-vehicle system that best fits this project[14]. Unfortunately, android auto does not support biometric authentication and Google Play on the emulator has not been updated to the most recent version which means we can not use google map services on it (Google Maps requires Google Play version 30 or above). For this, we can only make adjustments and design an Android mobile application instead. The application's main interface and the demonstration of one use case are presented in Figures 6.1 and 6.2 below:

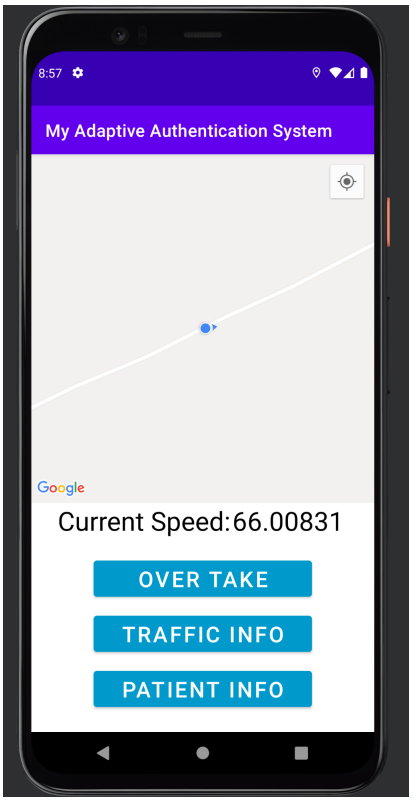

Figure 6.1: Main user interface

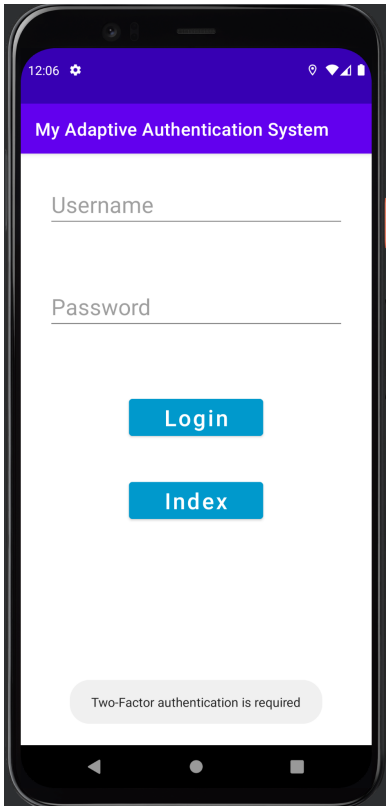

Figure 6.2: Showcase of the third scenario

6.1 Implementation that satisfies Confidentiality

As we mentioned at 5.1, this scenario is not affected by contextual factors, the focus of this implementation should be on designing the certificate transmission mechanism of different certificates among the four components between the server and the client as well as generating the necessary certificates. The structure of the transmission process is depicted in the diagram below.

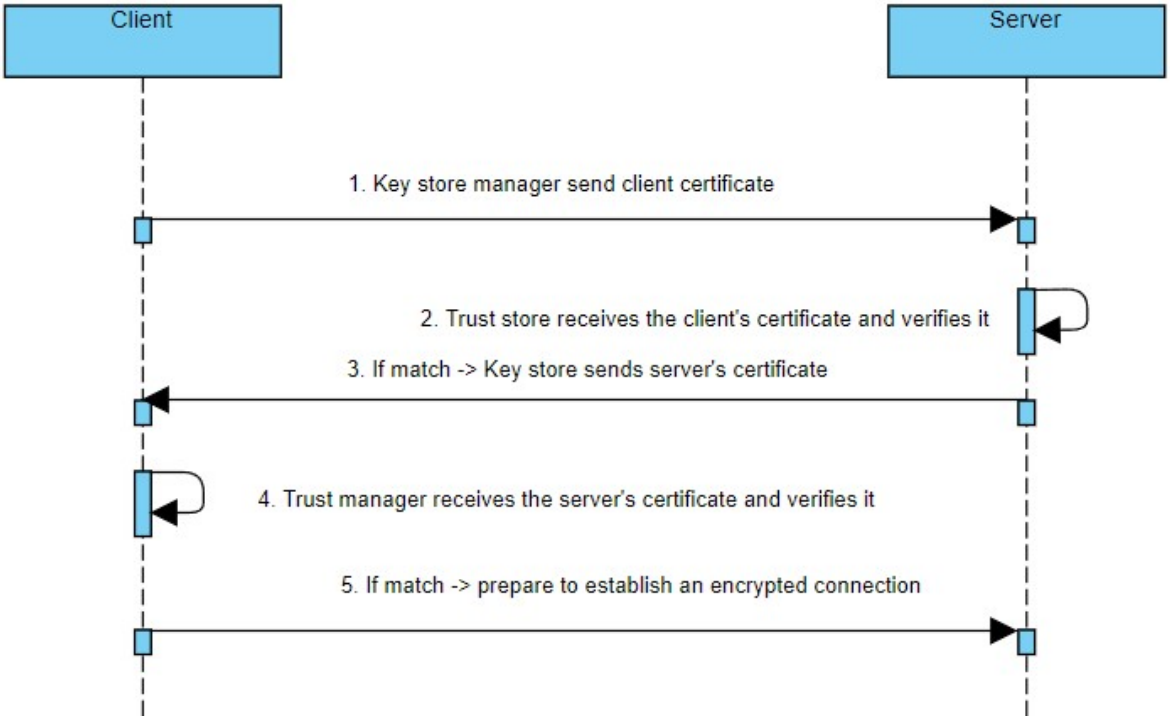

Figure 6.3: Transmission structure

6.1.1 The operations to generate CA-signed certificates:

Create root certificate (using command prompt):

1. Generate root certificate private key: openssl genrsa -des3 -out root.key 1024
2. Generate the root certificate signing request file: openssl x509 -req -in root-req.csr -out root-cert.cer -signkey root.key -CAcreateserial -days 365
3. Self-signed root certificate: openssl x509 -req -in root-req.csr -out root-cert.cer -signkey root.key -CAcreateserial -days 365
4. Export certificate in p12 format: openssl pkcs12 -export -clcerts -in root-cert.cer -inkey root.key -out root.p12

Create server side certificate (using command prompt):

1. Generate server key: openssl genrsa -des3 -out server-key.key 1024
2. Generate server request file: openssl req -new -out server-req.csr -key server-key.key

3. Generate server certificate (root certificate, rootkey, server key, server request file, these four generate client certificate): openssl x509 -req -in server-req.csr -out server-cert.cer -signkey server-key.key -CA root-cert.cer -CAkey root.key -CAcreateserial -days 365

4. Generate server-side p12 root certificate: openssl pkcs12 -export -clcerts -in server-cert.cer -inkey server-key.key -out server.p12

Create a client side certificate (using command prompt):

1. Generate client key: openssl genrsa -des3 -out client-key.key 1024

2. Generate client request file: openssl req -new -out client-req.csr -key client-key.key

3. Generate client certificate (root certificate, rootkey, client key, client request file four generate client certificate): openssl x509 -req -in client-req.csr -out client-cert.cer -signkey client-key.key -CA root-cert.cer -CAkey root.key -CAcreateserial -days 365

4. Generate client-side p12 root certificate: openssl pkcs12 -export -clcerts -in client-cert.cer -inkey client-key.key -out client.p12

6.1.2 Certificate base authentication Server-side design:

After we have both side certificates ready, we will put up a property file on the server to enable both ways SSL authentication. In this paper, we explore the use of two forms of SSL authentication certificates, although only the OpenSSL certificate chain will be used in our project, as we previously described.

OpenSSL : certificate chain

```
server.ssl.enabled=true
server.ssl.key-store-type=JKS
server.ssl.key-store=scenario_2_TrafficInfo/src/main/store/server.jks
server.ssl.key-store-password=961008
server.ssl.key-alias=1
server.ssl.trust-store=scenario_2_TrafficInfo/src/main/store/root.jks
server.ssl.trust-store-password=961008
server.ssl.trust-store-provider=SUN
server.ssl.trust-store-type=JKS
server.ssl.client-auth=need
```

Keystore : self-sign certificate

```
server.ssl.enabled=true
server.ssl.key-store-type=JKS
server.ssl.key-store=scenario_2_TrafficInfo/src/main/store/localhost.jks
server.ssl.key-store-password=961008
server.ssl.key-alias=server
server.ssl.trust-store=scenario_2_TrafficInfo/src/main/store/localhost.jks
server.ssl.trust-store-password=961008
server.ssl.trust-store-provider=SUN
server.ssl.trust-store-type=JKS
server.ssl.client-auth=need
```

We will enable both ways SSL authentication by setting ssl.client-auth to need, and store server-side certificate into Keystore, Keystore stores the certificate for server which is going to send to the client, store root certificate into trust store, trust store stores the certificate for the client which tells the server which client can be trusted.

As discussed in the previous section, we will simply construct one simple restful API to simulate commutation, which will provide an accepted HTTP status to the client if it is visited.

6.1.3 Certificate base authentication Client-side design:

1. Load server's certificate to trust store which tells the client which server can be trusted then create a trust management factory instance with default algorithm, finally Initialize the server trust store.

```
private TrustManagerFactory LoadServerCertificate() throws Exception {
    KeyStore trustStore = KeyStore.getInstance("PKCS12");
    trustStore.load(server certificate, password);
    TrustManagerFactory trustManagerFactory =
        TrustManagerFactory.getInstance(TrustManagerFactory
    .getDefaultAlgorithm());
    trustManagerFactory.init(trustStore);
    return trustManagerFactory;
}
```

2. Load client's certificate to key store which is going to send to the server then creates a key management factory instance with default algorithm, finally Initialize the client key store.

```
private KeyManagerFactory LoadClientCertificate() throws Exception {
    KeyStore keyStore = KeyStore.getInstance("PKCS12");
    keyStore.load(client certificate, password);
    KeyManagerFactory keyManagerFactory =
        KeyManagerFactory.getInstance(KeyManagerFactory
    .getDefaultAlgorithm());
    keyManagerFactory.init(keyStore,password);
    return keyManagerFactory;
}
```

3. Create HttpsURLConnection instance: conn and points to our server, initialize SSL context with key manager factory, and trust manager factory, this is the indispensable process before we can move to the final step, the SSL context will be utilized to compare the certificates before establishing an encrypted connection.

```
private void SSLConnection(){
        HttpsURLConnection conn = url.openConnection();
        SSLContext sslContext=SSLContext.getInstance("TLS");
        sslContext.init(LoadClientCertificate().getKeyManagers(),
            trustAllCerts, null);
}
```

4. Finally, load the SSL context to the instance we created so that we are able to start the connection.

We discovered that Android does not allow unregistered CA to publish certificates during the client-side implementation. So we have to create our own trust manager: trustAllCerts to trust all SSL certificates come from the server-side (overwrite the trust manager class and leave the functions empty), by doing so we can establish an encrypted connection using CA-signed certificate which sign by our own unregistered certificate authority root account.

6.2 Implementation that satisfies Performance

As we discussed in the earlier section 5.2, this scenario is affected by the speed and the traffic, based on whether the vehicle is moving and the traffic condition is good or bad, the client should decide which of the two authentication methods we have screened should be performed (since we were unable to obtain traffic information from any google services, we will not implement the mechanism which makes the traffic factor affects the authentication process for this scenario). Following verification, the client sends the server the built-in license plate number and password to acquire the token, which may then be used to obtain distance information. The operation sequence diagram is shown below.

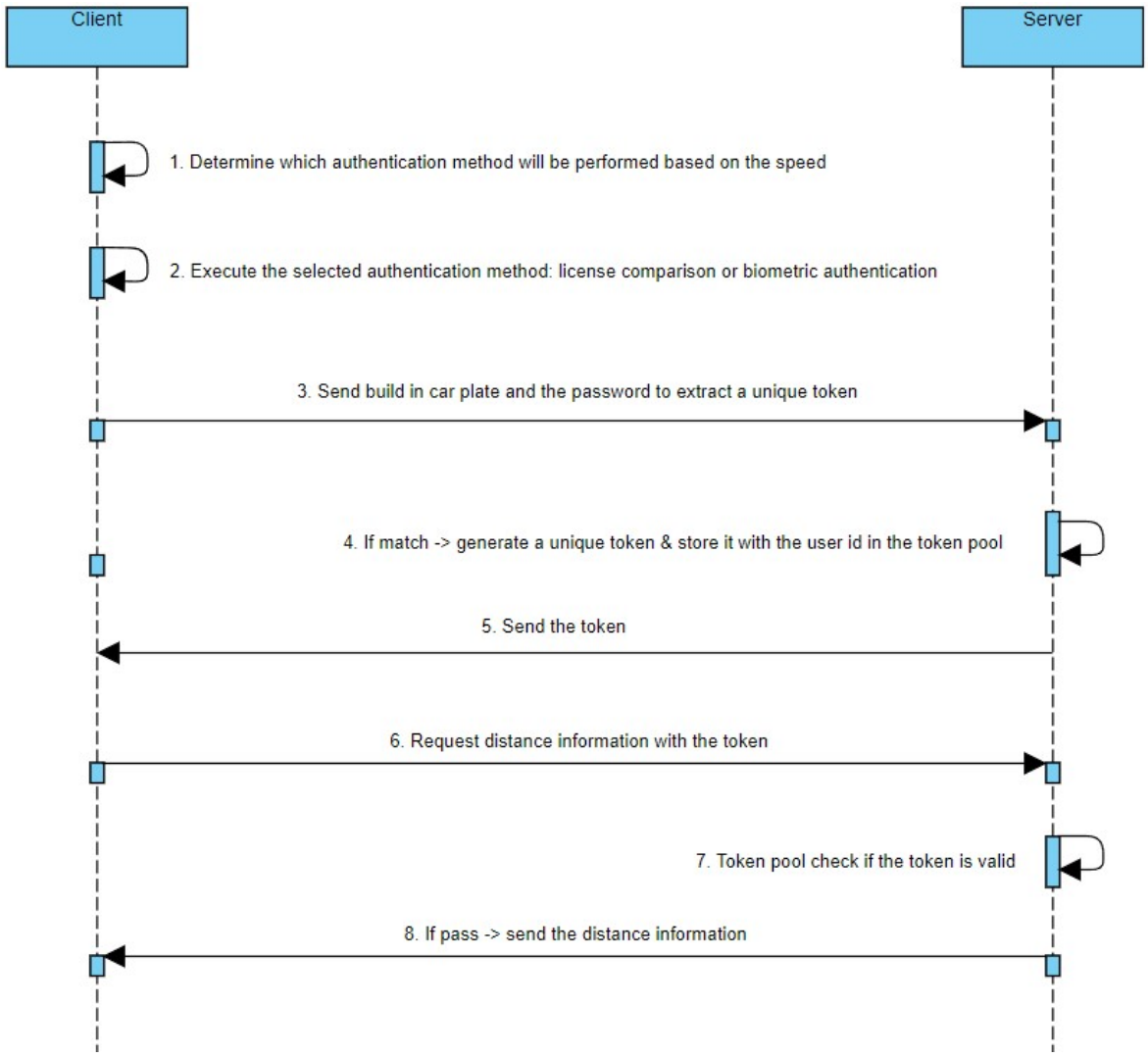

Figure 6.4: operation diagram

6.2.1 Client-side design

The client side for this scenario will be constructed based on the pseudocode 1 below, the user first reacts with a overtake function which will obtain the speed of the current vehicle, then make a decision about which authentication method will be used based on the moving speed, if the vehicle is moving then the action should happen quickly since performance has the highest priority, in this case, the license comparison will be invoked; otherwise, the biometric activity will be selected. Once the user passes the authentication, the gettoken function should be called automatically to submit the user's car plate and password to the server, which will verify the user and provide a unique token. Finally, the user will be able to access distance information using the token.

Pseudocode 1 Logic design

```
1: if the vehicle is moving then
2:     license comparison();
3:     if pass then
4:         gettoken();                ▷ Send the credential to the server to get a unique token
5:         if pass then
6:             ifuploadssuccess();    ▷ Upload locations and receive distance with the token
7:         end if
8:     end if
9: else
10:    biometric authentication();
11:    if pass then
12:        gettoken();                ▷ Send the credential to the server to get a unique token
13:        if pass then
14:            ifuploadssuccess();    ▷ Upload locations and receive distance with the token
15:        end if
16:    end if
17: end if
```

The functions will be used in the order below to put the pseudocode into practice:

1. overtake(): define a listener that responds to location updates in order to obtain the current speed
2. decisionmaking(): take the speed as an argument then decide which authentication methods should be performed
3. licenseComparison(): compare car plate numbers with built-in information to determine if the user can pass the authentication(this refers to the License based authentication described below)
4. biometricactivity(): biometric authentication to determine if the user can pass the authentication(this refers to the Biometric based authentication described below)
5. gettoken(): use a RestTemplate object to submit the built-in car plate number with the corresponding password(use the getdatabydevice() below to query this data) to the server to retrieve a unique token
6. ifuploadsuccess(): set header with the token and use a RestTemplate object to communicate the location information to the server then receive distance information

We will need to build license-based authentication and biometric-based authentication to enable the mechanism described above.

License based authentication

To store and query data, we created a database helper class. The following functions will be utilized within the DBhelper class:

1. insertUserInfo(): insert a built in car plate and password into the Android local database
2. update(): update the built in car plate and password into the Android local database
3. getdatabydevice(): query data by car plate

The final stage in this authentication will be to use the provided function to match the current license plate number with the built-in one.

Biometric based authentication

To enable biometric authentication on the Android system, we will first construct a BiometricManager object, which will assist us in determining whether or not the current system contains a biological information collection component.

```
BiometricManager biometricManager = BiometricManager.from(this);
    switch (biometricManager.canAuthenticate()) {
        case // user can use biometric sensor
        case // device does not have fingerprint sensor
        case // biometric sensor unavailable
        case // no finger print saved in this device
    }
```

Then build an executor object and load it to biometricPrompt, this will return the authentication result.

```
Executor executor = ContextCompat.getMainExecutor(this); //executor gives
    the result of the authentication
    BiometricPrompt biometricPrompt = new
        BiometricPrompt(BiometricAuthenticationActivity.this, executor,
        new BiometricPrompt.AuthenticationCallback() {
    // based on the results of authentication do some action
    }
```

We noticed that a connection refused error occurs when we use restTemplate.postForObject ("http://localhost:8080/someurl") to submit data to the server. After some research, we realize that we are using an Android emulator to refer to our localhost on our system, but 127.0.0.1 refers to the emulator, not our local machine, so we have to use http://10.0.2.2:8080/ to bridge to our local machine instead.

6.2.2 Server-side design

Our server holds two core functionalities:

- Verification

- Upload and return distance information

Verification

To perform a server-side verification we will create our own custom annotation, a custom annotation will create a new layer between incoming requests and the target API. This new layer intercepts incoming requests and runs checks to decide whether or not they should be allowed to pass. To achieve this the following steps were followed:

1. Create a unique token pool

At first step we need create a token pool class that holds a HashMap object, the hashmap should store the car palate with a unique UUID. This class holds several helper functions:

- login(): store car plate with the token into the hashmap
- generateToken(): generate a unique UUID
- getUserIdByToken(): find token by input car plate number
- getTokenByUserId(): find car plate number by input token
- containsToken() : check if the hashmap contains an input token

2. Custom annotation (Aspect Orient Programming)

We will construct a RestrictUserAccess interface first, which defines the default user level required to access APIs restricted by custom annotations. Then we need a RestrictUserAccessAspect class, it helps us to restrict the users who can access our servers. This class only holds one function:

- restrictUserAccess(ProceedingJoinPoint joinPoint): determine if the current user has the privilege to visit the target API.

```
@Around("@annotation(service.scenario_3.access.RestrictUserAccess)")
    public Object restrictUserAccess(ProceedingJoinPoint joinPoint){
        MethodSignature signature = joinPoint.getSignature();
        Method method = signature.getMethod();
        /**
        collect the target API's privilege requirements, treat as default
            level if not present.
        */
        UserLevel userLevel =
            method.getAnnotation(RestrictUserAccess.class).requiredLevel();
        try {
            //capture the token from the first argument of the request
            String token = (String) joinPoint.getArgs()[0];
            //ask token pool to check whether the user is logged in
            String trustdvice=tokenPool.getUserIdByToken(token);
            User user=userRepository.findByUserBytrustdvice(trustdvice);
            if(user==null){
                //the user did not log in
            } else if(user.getUserLevel() != userLevel){
```

```
                //the user does not have the privilege to visit the target API
            } else {
                //allow the request to pass to our API
            }
        } catch (Exception e) {
            return e.getMessage();
        }
    }
```

To compare user privilege, we acquire the token from the request header, then use the token pool class's helper method to discover the car plate by the input token and see if the account has access to the target API. In order to use the helper function, the token pool class must be auto wired to RestrictUserAccess class.

The token in the request header was received from the first argument of the join point,and the user level required to access the target API was collected from the join point's getAnnotation method in a method object.

To enable the restriction, simply put @RestrictUserAccess(requiredLevel = UserLevel.userlevel) on the target API.

```
@RestrictUserAccess(requiredLevel = UserLevel.admin)
@RequestMapping(value="/exampleAPI",method= RequestMethod.POST)
@ResponseBody
public ResponseEntity<Double> exampleAPI(@RequestHeader("token") token){}
```

Upload and return distance information

The following procedures were required to upload and return the distance to the client:

1. restful API: receives the positions of the ambulance and target vehicle
2. distanceCalculation(): determine the distance between the two places in meters

We created a restful API that receives the positions of the ambulance and target vehicle, use a helper function to determine the distance between the two places in meters, then return the result distance information back to the client. An admin privilege is required to access this API.

6.3 Implementation that satisfies Both Confidentiality and Usability

In this part, we should construct two authentication methods(username and password based authentication, biometric based authentication) that will be applied in two-factor authentication, and

link them together using the approach described in section 5.3. We use the time factor to determine which biometrics should be captured when implementing biometric authentication. However, since we were unable to select specified biometric authentication techniques, our system for this case will not be adaptive over time, as Android exclusively employs fingerprint authentication.

The logic of two-factor authentication was built in the order illustrated in sequence diagram 6.5 below. The client sends the user name and password to the server first so the user can be verified on the server-side and receive a returned token, then verify the user's biometric information on the client-side, and finally use the token to access the patient information.

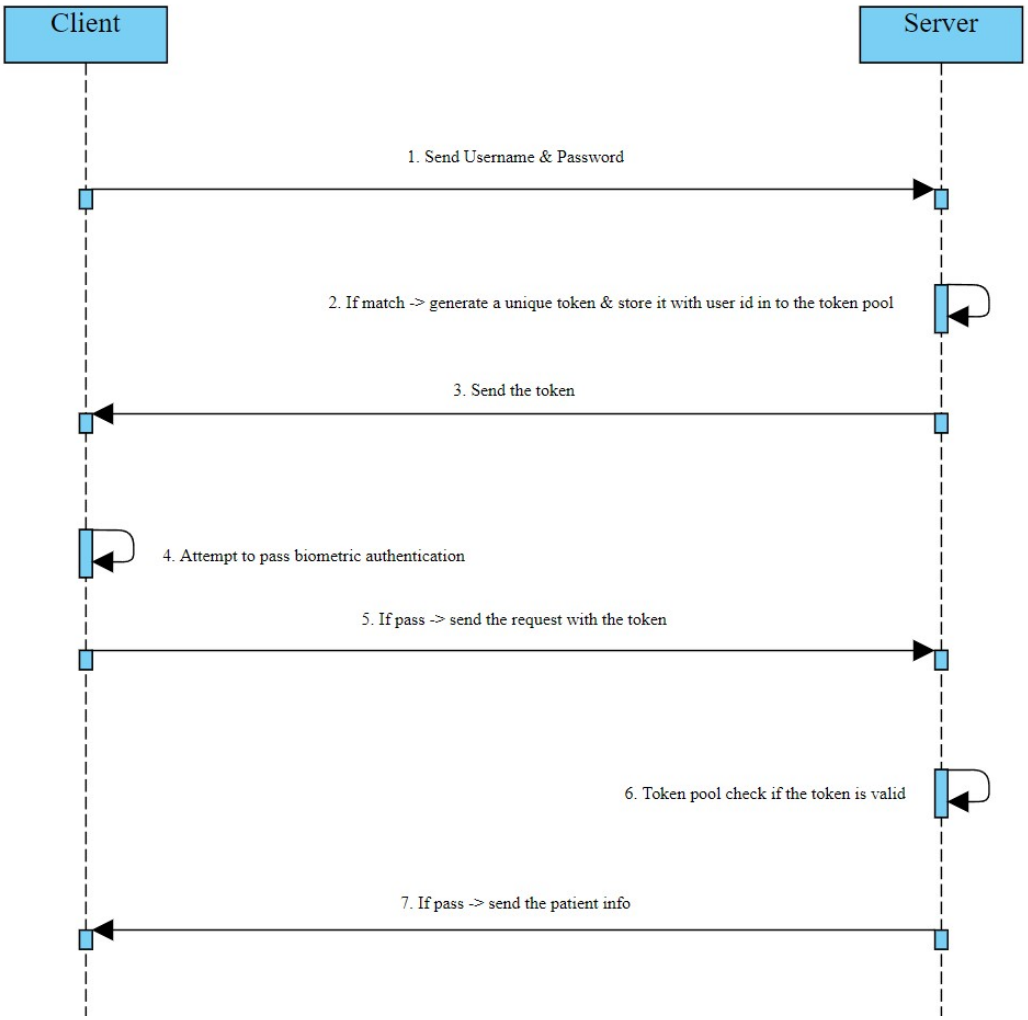

Figure 6.5: Two-factor based authentication

6.3.1 Client-side design

Our client hold two authentication methods

- Username and password based authentication
- Biometric based authentication

Username and password based authentication design

We publish the input username and password to a resttemplate object, which then receives and briefly stores a unique token. An intent object will be used to pass the token to the following activity after it has been acquired.

```
Intent intent = new
    Intent(getApplicationContext(),BiometricAuthenticationActivity.class);
intent.putExtra("token", token);
startActivity(intent);
```

Biometric based authentication design

The structure of biometric-based authentication in this process will be identical to that described in section 6.2.1, with the addition of a time factor. We'll provide a helper function to determine whether it's day or night so that the developers who have the ability to select specific biometric authentication methods can make the system adaptive base on time in further development (fingerprint during the nighttime, face scan during the daytime).

After passing the biometric authentication our system should also be able to use the token from the previous authentication activity to visit the server, hence an additional helper function is required as below:

1. isNight(): assess whether it is day or night
2. ifsucess(): set header with the token and use a RestTemplate object to communicate to the server then receive patient information

6.3.2 Server-side design

Our server should receive the username and password from the client, then check the identification of the user, the verification process will be the same as we described in section 6.2.2, so we won't go through it again here.

Chapter 7: Evaluation of the performance

The most important component of a mobile app's success is its performance, it is difficult for an app to retain users if it performs poorly. In this chapter, we will run performance tests for each scenario to evaluate the corresponding CPU usage, memory usage, and battery consumption, then make assumptions about the reasons that caused the results.

A mobile performance profiling tool called Apptim was involved to test our application[15].

7.1 Performance evaluation for scenario 1: acquire road traffic information

7.1.1 CPU usage

The CPU usage during the SSL handshake increased to nearly 25 percent, after the handshake was completed it then almost dropped to 0, later the application received a message from the server which cost around 20 percent of CPU usage.

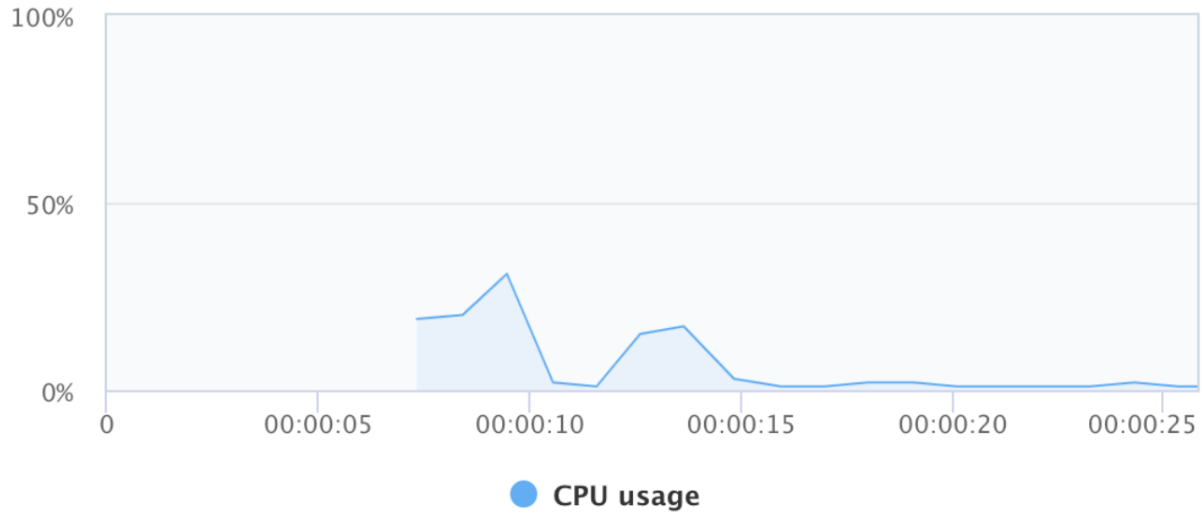

Figure 7.1: CPU usage to acquire road traffic information

7.1.2 Memory usage

The software required a substantial amount of memory during encrypted communication; it is obvious that the memory was expanded to over 75 MB and then slightly decreased to almost 50

MB before stabilizing.

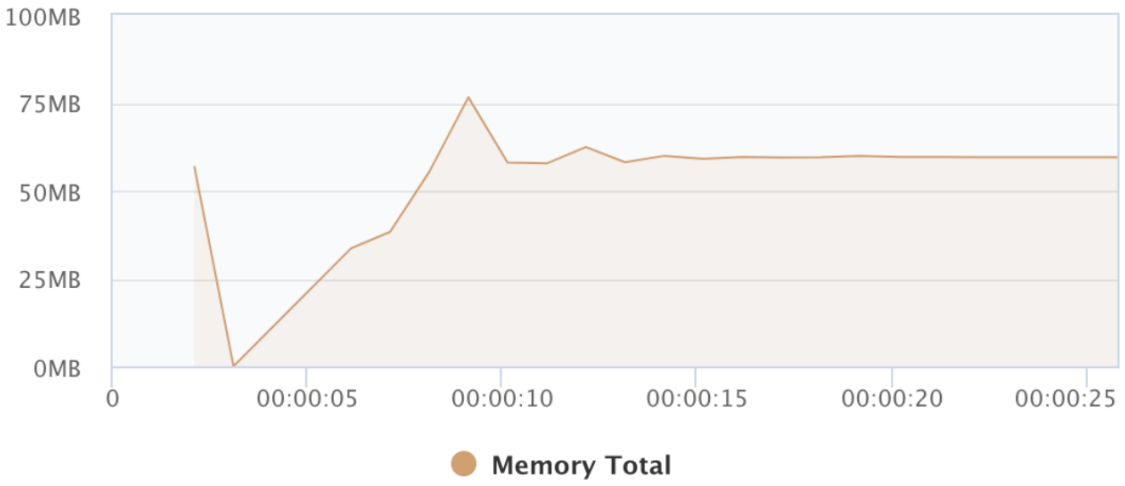

Figure 7.2: Memory usage to acquire road traffic information

7.1.3 Battery consumption

Figure 7.3 shows an erratic power consumption situation. This may be caused by constant interaction between client and server.

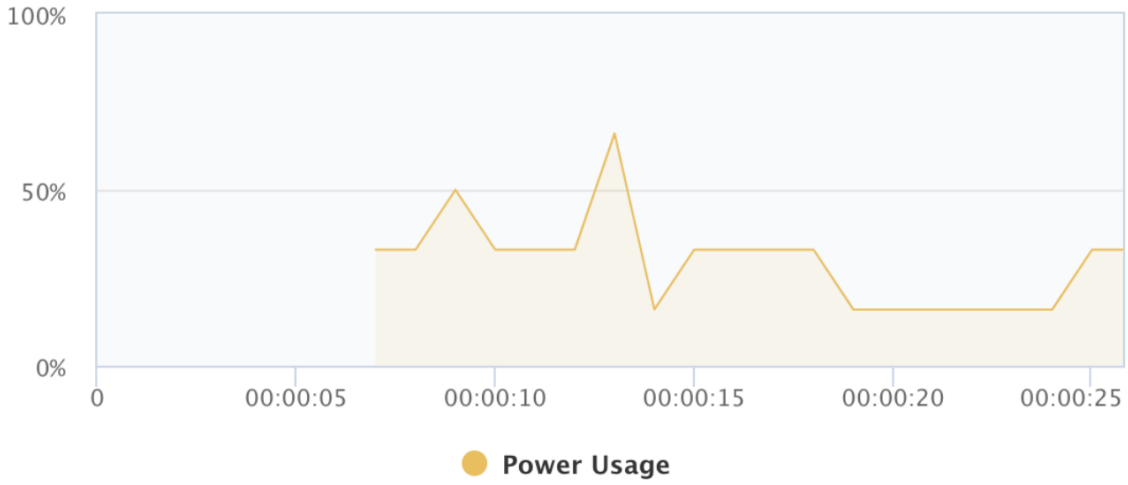

Figure 7.3: Battery consumption to acquire road traffic information

7.2 Performance evaluation for scenario 2: overtake another vehicle

7.2.1 CPU usage

The CPU usage increased to almost 50 percent at the beginning, which indicates that the software requires a lot of CPU efficiency during the initial execution stage of the authentication process in order to collect speed information and select the authentication method. After the data collection and decision-making are completed the CPU usage then slowly decreased.

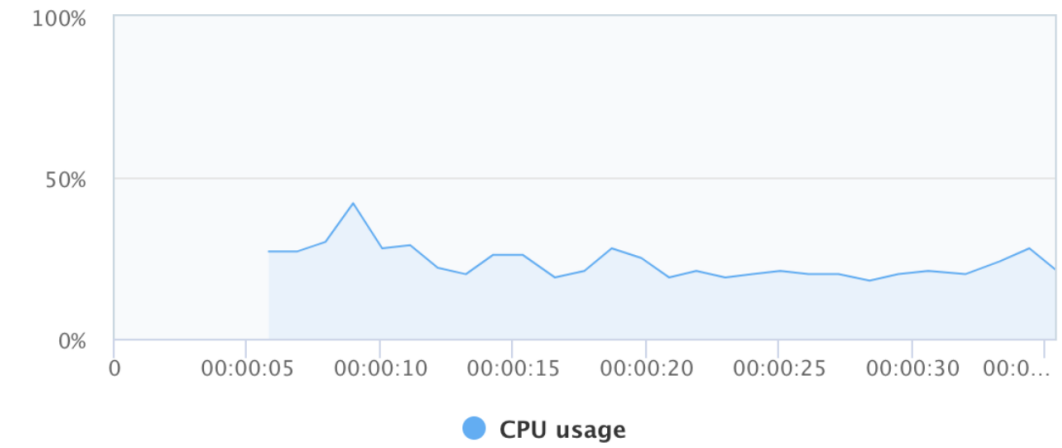

Figure 7.4: CPU usage to overtake another vehicle

7.2.2 Memory usage

The same goes for memory usage, as we can see from Figure 7.5 the memory usage non-linearly increased to about 75MB at beginning of the process, as the software needs to consume a large amount of memories during the initial execution stage of the authentication process in order to collect speed information and select the authentication method. After the data collection and decision-making are completed, the CPU usage first slowly dropped to 70MB then stabilized.

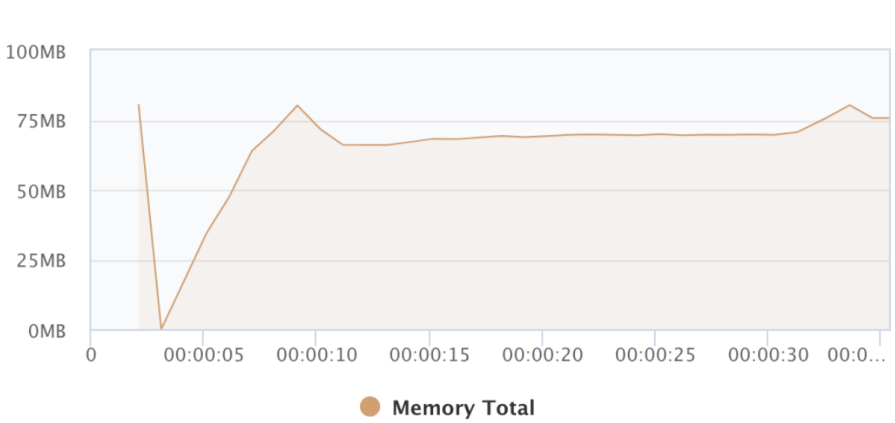

Figure 7.5: Memory usage to overtake another vehicle

7.2.3 Battery consumption

In the early authentication process, the battery consumption reached close to 40 percent and was maintained for a period of time since the data collection and decision making are costly actions, after the initial process was completed the battery consumption dropped down to nearly 20 percent for a short time. Then the application was trying to connect to the server which cost battery consumption back to almost 40 percent.

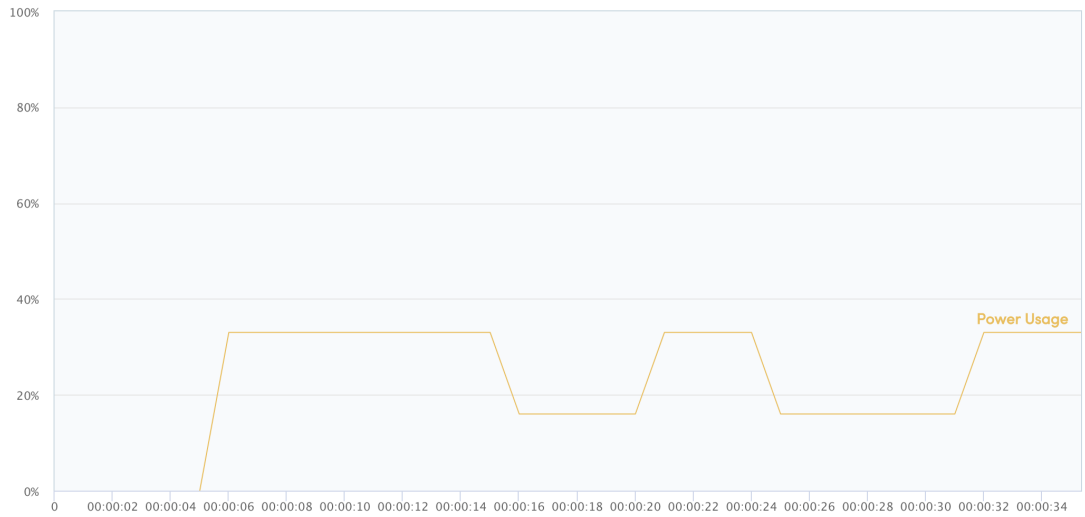

Figure 7.6: Battery consumption to overtake another vehicle

7.3 Performance evaluation for scenario 3: access patient's information

7.3.1 CPU usage

The conversion between the two authentications may produce undulating CPU consumption when multiple authentications are used as Figure 7.7 shows below.

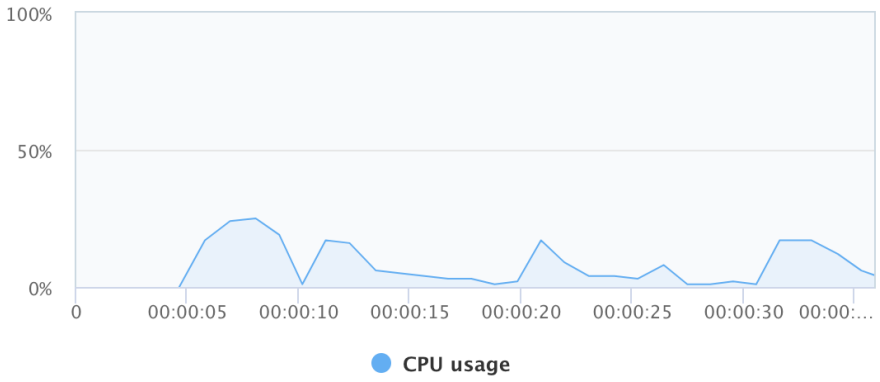

Figure 7.7: CPU usage to access patient's information

7.3.2 Memory usage

We assume that multiple-step authentication does not cause an erratic memory usage as Figure 7.8 presents a rapidly increasing then gradually flattening memory usage.

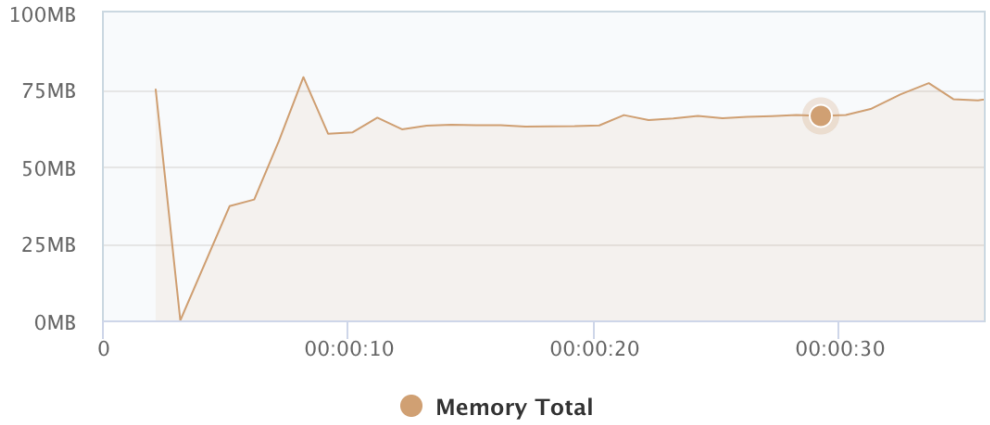

Figure 7.8: Memory usage to access patient's information

7.3.3 Battery consumption

Figure 7.9 shows that multiple-step authentication may cause an erratic power consumption situation.

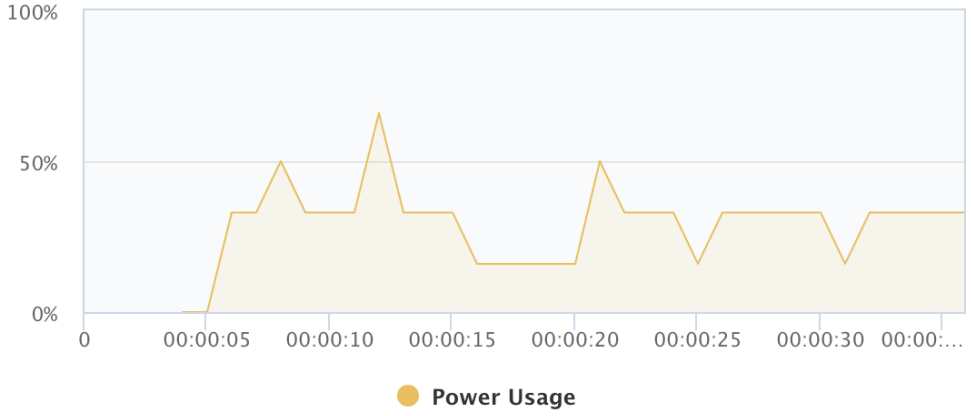

Figure 7.9: Battery consumption to access patient's information

Summary

Based on the information presented above, it is safe to conclude that overtaking another vehicle necessitates the most expensive authentication procedure (consume massive CPU and memory usage). Obtaining road traffic information and patient information results in irregular power usage, which in some situations may result in excessive power consumption.

Chapter 8: **Conclusions and further works**

In this project, we present an Android application to simulate three authentication scenarios, for each scenario we described the challenges that adaptive authentication may encounter and provided a solution based on the contextual factors and requirements.

Several restrictions were discovered during the development of this application and we come out with the corresponding solutions as the list below:

1. For the second scenario, we are unable to acquire traffic data in order to improve the accuracy of the authentication system's adaptation.

Solution:
The developer should contact the google team in order to gain permission to access traffic data.

2. For the third scenario, we are unable to define which biometric authentication will be performed to make the authentication system adaptive base on the time factor.

Solution:
the developer should consider setting authorized authenticators to weak types to force the system to reduce the strength of the biometric authentication methods

3. The adaptive authentication system we present can only handle a limited number of situations.

Solution:
The developers should consider a wider range of scenarios (use cases for this application). This is critical since it is impossible to accurately forecast real-world application circumstances. We can only expand our system's flexibility by making a huge number of assumptions. The more possibilities we evaluate, the more adaptable the system becomes.

In conclusion, although our initial solution has limitations, it can still provide guidance to the third-party developers on how to implement a mobile adaptive authentication in their own system.

Acknowledgements

I would like to thank University College Dublin, School of Computer Science for their support during the four years of my undergraduate course. I extremely appreciate my supervisors Liliana Pasquale and Alzubair Hassan, for help and supervision on this project.

I also want to thank my classmate WeiTong Chen for proofreading and grammar check.

Bibliography

1. Hassan, A., Nuseibeh, B. & Pasquale, L. *Engineering Adaptive Authentication* in *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)* (2021), 275–280.
2. Abu Bakar, K. A. & Haron, G. R. *Adaptive authentication based on analysis of user behavior* in *2014 Science and Information Conference* (2014), 601–606.
3. Katsini, C., Belk, M., Fidas, C., Avouris, N. & Samaras, G. *Security and usability in knowledge-based user authentication: A review* in *Proceedings of the 20th Pan-Hellenic conference on informatics* (2016), 1–6.
4. Bakar, K. A. A. & Haron, G. R. *Adaptive authentication: Issues and challenges* in *2013 World Congress on Computer and Information Technology (WCCIT)* (2013), 1–6.
5. Arias-Cabarcos, P., Krupitzer, C. & Becker, C. A survey on adaptive authentication. *ACM Computing Surveys (CSUR)* **52,** 1–30 (2019).
6. Daud, N. I., Haron, G. R. & Othman, S. S. S. *Adaptive authentication: Implementing random canvas fingerprinting as user attributes factor* in *2017 IEEE Symposium on Computer Applications Industrial Electronics (ISCAIE)* (2017), 152–156.
7. Daud, N. I., Haron, G. R. & Din, D. *Adaptive Authentication to determine login attempt penalty from multiple input sources* in *2019 IEEE Conference on Application, Information and Network Security (AINS)* (2019), 1–5.
8. Hulsebosch, R. J., Bargh, M. S., Lenzini, G., Ebben, P. W. G. & Iacob, S. M. Context Sensitive Adaptive Authentication. *Lecture Notes in Computer Science Smart Sensing and Context,* 93–109.
9. Seifert, J., Luca, A. D., Conradi, B. & Hussmann, H. TreasurePhone: Context-Sensitive User Data Protection on Mobile Phones. *Lecture Notes in Computer Science Pervasive Computing,* 130–137 (2010).
10. Gebrie, M. T. & Abie, H. Risk-based adaptive authentication for internet of things in smart home eHealth. *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings* (2017).
11. Valero, J. J. *et al.* Improving the Security and QoE in Mobile Devices through an Intelligent and Adaptive Continuous Authentication System. *Sensors* **18,** 3769 (2018).
12. Riva, O., Qin, C., Strauss, K. & Lymberopoulos, D. *Progressive authentication: deciding when to authenticate on mobile phones* in *21st {USENIX} Security Symposium ({USENIX} Security 12)* (2012), 301–316.
13. IBM. *keytool - Key and Certificate Management Tool* https://www.ibm.com/docs/en/sdk-java-technology/7?topic=keytool-key-certificate-management-tool (2022).
14. *Android Auto* https://developer.android.com/cars (2022).
15. *Apptim* https://www.apptim.com/mobile-performance-profiling.