A Secure and ID Privacy-Preserving Distributed Data Collection (SPDC) framework for Internet of Things Application

A thesis submitted to the University of Manchester for the degree of
Doctor of Philosophy
in the Faculty of Science and Engineering

2022

By
Tahani Hamad Aljohani
Department of Computer Science

Contents

Word Count: 52,501

List of figures

List of tables

Abstract

We have witnessed a huge demand for remote provisioning of medical and healthcare services, where patients can access healthcare services via an application on their smartphones. With the rapid advance of IoT (Internet of Things) technologies, patients' medical and health conditions will be monitored remotely in real-time via wearable devices or sensors worn by the patients. A core functional component in such a patient health monitoring system is the collection of a patient's health data using wearable devices anywhere and anytime. Ensuring the security of data and identity (ID) privacy of the patient during this data collection process is paramount. This thesis investigates how to facilitate remote collection of a patient's data anywhere and anytime while protecting the confidentiality of the data being collected and preserving the privacy of the patient's ID. The investigation is carried out in a context in which the data collection service is provided by a third-party service provider. This opens up a number of security and ID privacy issues. To this end, the thesis has presented a novel framework for Secure and ID-Preserving distributed Data Collection (SPDC). In designing this framework, we have addressed three open questions which are identified based on our critical analysis of existing solutions: (i) how to support authorized use of the data collection service by a patient, that is provided by the third-party service provider without revealing the patient's real ID (ii) how to prevent the inference of a patient's ID from his/her contextual information and data collection service usage patterns and (iii) how to minimize processing load imposed on the end server thus reducing the risk of creating a performance bottleneck and supporting scalability. In addressing these questions, the thesis makes the following novel contributions. First, it presents a novel architecture for the framework which supports the use of a distributed set of data collection servers owned by different service providers and the healthcare provider. Second, the patient can select one server to be the home server, and select a number of servers to be the foreign servers. Third, the SPDC allows the patient to access any data collection servers using certificates generated by the patient. Fourth, the SPDC allows the patient to upload to the foreign servers using a swarming algorithm to hide the uploading pattern, the home server is responsible for collecting the patient's data from the foreign servers and sending them to the healthcare provider. Fifth, the SPDC proposes a method for efficient verification of each request (uploading) by the patient without searching the server's database for the verification key to verify the request. The framework has been analyzed using a bench-marking tool and evaluated using queuing theory. The evaluation results indicate an efficient performance when the number of servers increases.

Declaration of originality

I hereby confirm that no portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright statement

Dedication

In the name of Allah, the most Gracious, the most Merciful
”Your Lord has ordered you to worship none except Him, and to be good to your parents. If either or both of them attain old age with you, do not say: ”Fie on you”, nor rebuke them, but speak to them with words of respect (23) And lower to them the wing of humbleness out of mercy and say: ’My Lord, be merciful to them, as they raised me since I was little.’ (24)” [AL-ISRA, the Holy Quran].

To Mum and Dad.

Acknowledgements

I thank Allah for all his blessings; without his guidance and success I would not have completed and presented my PhD thesis.

I would like to express my sincere appreciation and gratitude to my parents who have always encouraged me and my siblings to expand our knowledge and have devoted their lives to us.

I would like to thank my daughters (Deem and Lena) for being patient while their Mom was busy.

I would like to thank my supervisor Dr Ning Zhang for her valuable advice throughout my PhD study. Dr Ning is such a great, thoughtful, and patient leader.

Chapter 1

Introduction

1.1 Research Context

The Internet of Things (IoT) can be defined as the paradigm of connecting smart things (e.g. sensors, devices) together by means of information and communication technologies to build intelligent systems and services to obtain required information. The smart things can sense the surrounding environment and communicate with each other to exchange information. These features (i.e. sensing and communicating) facilitate many emerging attractive applications. One of these applications is Patient Health Monitoring (PHM) systems. PHM involves the use of mobile computing and wireless communication technologies to regularly collect data from a patient for the purpose of analyzing the patient's health and making health-related decisions.

Many countries are suffering from an increasing number of medical patients. Some patients suffer from chronic diseases such as heart attacks and need continuous health monitoring. In some cases, patients have to be under 24 hours of intensive care. This requires doctors to stay connected with the patients face-to-face. In addition, the healthcare provider should provide sufficient facilities (e.g. beds) for patients, which requires a large budget. Presently, with the rise of IoT and wearable devices, several health systems have developed such as Patient Health Monitoring (PHM) systems. These systems have improved the quality of patients' life. With PHM systems, physicians can treat more patients at any time without face-to-face interaction. Patients can stay connected with health providers as needed. The uses of PHM systems have reduced medical costs and improved the quality of care by providing health services anywhere and anytime.

A typical PHM system, as seen in Figure 1.1, consists of body sensors and a mobile or fixed device at the patient's end (e.g. home) and remote servers at the healthcare provider's end. The body sensors worn by the patient are connected wirelessly to the device that is, in turn, connected to the servers via wireless and/or wired networks. The health-related data (e.g. heart rate, blood pressure, etc) collected (or sensed) by the body sensors are sent to the device, which are then delivered to the healthcare provider for further analysis and decision making.

The operations performed by a PHM system are typically of three stages: data collection, data analysis (i.e., the analysis of the collected data), and decision making (based on the

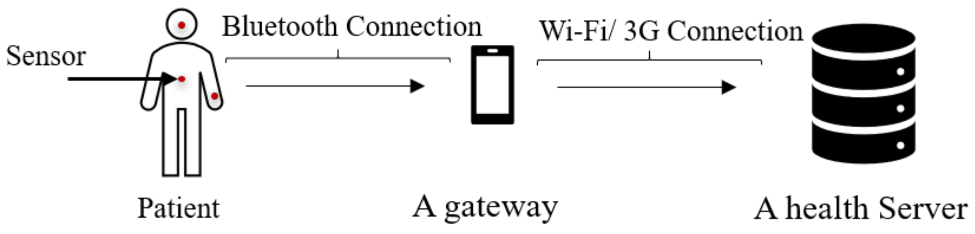


Figure 1.1: A typical PHM system

outcome of the data analysis). The data collection stage is crucial to the correct running of a PHM system, as the correctness of the analysis and decision making are dependent on the correctness of the data collected.

It is expected that future PHM systems will be built on infrastructure owned by a third-party service provider which has more resourceful storage and data processing capabilities, such as Microsoft. This is because on-premises infrastructures, on which most healthcare providers rely today, might not be able to handle the volume of data generated from wearable devices. It is estimated that by 2021 more than 222 million wearable devices may be poured onto the market and three in five patients may use remote monitoring services. These devices can generate data at high frequencies (e.g. every 5 minutes), generating massive volumes of data. We expect that patient data will be collected by the service provider [1], [2], which will bring up security and privacy threats as follows.

- Authentication threats: An adversary may try to impersonate a patient to gain unauthorised access to the patient's data.
- Data authenticity threats: The patient's data may be delayed, replayed, or even modified during transit. In addition, the service provider may try to generate data on behalf of the patient.
- Data confidentiality threats: Threats that may happen because the patient's data is not protected during transit.
- Patient's ID privacy threats: If the patient is using the same ID even if it is an artificial one when communicating with the service provider, the latter might collate the information known about the patient with information elsewhere on the Internet to find the real identity of the patient. Moreover, an external adversary may link multiple sessions to the same patient.
- Inference threat: By using the same service provider each time to upload health data, a threat known as an inference threat [3] may occur. Such a threat occurs when an unauthorized entity can use some sensitive attributes such as the pattern of communication (i.e., how many times per day a patient uploads data to the service provider) to link multiple transactions to the same patient. Consider this example. A patient uploads some data (such as blood pressure) at 1 pm every day and other data (such as heart rhythm) every 10 minutes. Then, even if that patient uses different pseudonym in each transaction, over time and by observing the upload pattern (i.e., 1 pm and every 10 minutes), the service provider may infer that all the data uploaded at this time belongs to the same patient.

In this thesis we designed an innovative framework called A Secure and ID Privacy-Preserving Distributed Data Collection(SPDC) to prevent a service provider from tracking the patient by observing communication patterns. The SPDC allows the patient to use different service providers (data collection servers) instead of one. The patient selects one of the servers to be the home server, while a number of servers to be the foreign servers. The patient randomly selects a foreign server each time for uploading. This is to disguise the patient by hiding the pattern of the uploading. In addition, for each server, the patient has a temporal ID and for each session with the same server, the patient has a fresh ID. By using multiple data collection servers and different IDs, we protect the patient against the inference threat. To protect against authentication threats, the patient should use pseudonym certificates to access service providers. The certificates are generated by the patient and signed blindly by the home server. To protect against the data authenticity and confidentiality threats, the patient should encrypt and sign the data before uploading. The SPDC framework has been analyzed using a benchmarking tool and evaluated using queuing theory. The evaluation results indicate an efficient performance when the number of servers increases.

1.2 Research Motivation and Challenges

By investigating current solutions for health and non-health data collection systems, it can be concluded that: (1) some existing solutions do address a scalable architecture but neglect both security concerns and privacy concerns, (2) some existing solutions either address the security concerns or privacy concerns, but not both, and (3) some existing solutions have neglected protecting patients' ID by hiding the access pattern. These three observations have motivated the research reported in this thesis.

To design a secure, ID privacy-preserving, and efficient data collection solutions, the following challenges should be addressed:

- How can the patient access these service providers anonymously and at the same time each service provider ensures that the patient is an authorized one?

- The patient has to conduct multiple uploads with the same service provider. However, the patient is not allowed to use the same artificial ID for each upload. This is to prevent linking multiple sessions by an external entity to the same patient. Now, how to allow the patient to use multiple artificial IDs, one for each transaction (uploading) and allow (only) the service provider which the patient is uploading with to link these IDs to the same patient?

- How is it possible to link patient data, which is scattered across service providers, to one account? In other words, which entity will link the patient's data, given that for each service provider the patient is using different artificial IDs?

- How and which entity is going to generate artificial IDs and credentials for each patient to access service providers?

- When the patient uploads the data on a service provider, how can we ensure that the service provider will not be able to generate data on behalf of the patient?

1.3 Research Aim and Objectives

The aim of this research is to investigate the state-of-the-art ways to design a secure, ID privacy-preserving, and efficient means to collect data from mobile patients. This aim is supported by the following objectives.

- To investigate and analyse PHM systems so as to (1) specify design requirements for a PHM data collection system, and (2) drive a generic use case scenario.
- To analyze the use case scenario so as to (1) specify design requirements for a PHM data collection system, and (2) identify a number of security and ID privacy threats.
- To identify, analyze and specify a set of requirements for a secure, ID privacy-preserving, and efficient solution to facilitate data collection from mobile patients anywhere and anytime.
- To investigate and critically analyze the state-of-the-art data collection solutions in the context of PHM so as to identify areas for improvements (i.e. knowledge gaps).
- To design a framework that satisfies the requirements and addresses the knowledge gaps. This framework consists of a system architecture, methods, and a set of protocols.
- To analyze the security of the protocols designed, the analysis should include informal analysis against the security requirements and security threats.
- To evaluate the performance of the designed protocols and architecture.
- To publish the findings.

1.4 Research Methodology

The research methodology used for this thesis consists of four key components: a literature survey and critical analysis of the related work, architectural design, protocol designs, and analysis and evaluation of the designed protocols.

1.4.1 Literature Review

The first task carried out in this research was to study the relevant literature in the area of interest: PHM systems and their operations. The purpose was to learn the generic architecture of the PHM system, how PHM works, and its operations. We found that the most critical operation in the PHM system is the data collection stage. Therefore, we focused on designing a data collection system for PHM applications. To do so we designed a generic use case scenario based on the use case scenarios found in the literature. This led us to identify the design requirements (including functional, security, ID privacy-preserving, and efficiency requirements). Using the design requirements, we critically analyzed the existing data collection solution for health or non-health applications. Although this is the initial research, it was apparent that there were few existing works on preserving patients' privacy in an efficient and scalable manner. It was discovered that existing solutions can be divided into (i) centralized data collection systems and (ii) distributed data collection systems. Solutions in (i) largely focused on how to protect the confidentiality of the health data through encryption and how to preserve the patient's ID by using a pseudonym. Solutions in (ii) focused on scalable architecture and neglected the security and privacy issues. Through critical reading and analysis of the centralized and distributed data collection solutions, a gap was identified. The gap was that no solution combined all design requirements (i.e., secure, ID privacy-preserving, and efficiency requirements). This also led us to our first system architecture. The system architecture consists of a number of data collection servers owned by different service providers which are selected independently by the patient to upload his/her health data. On top of this new architecture, we identified several security and performance issues. This led us to the first novel contribution of the thesis which is the SPDC framework. The literature was regularly reviewed throughout the duration of this research, with newly published work taken into consideration where necessary.

1.4.2 Framework Designs

After designing the First Data Collection (FDC) system, which consists of multiple data collection servers, we analyzed the FDC system to check if it satisfied the data collection design requirements. We discovered that the FDC system suffered from several security and performance issues. This stage of our work led us to design our innovative framework, which is called the SPDC. The SPDC's architecture consists of the same entities of the FDC system. However, in SPDC each patient has to select a number of servers and assign one of them as the home server, and the other servers as foreign servers. The home server is the anonymous linkage point for the patient, as it links the patient's data which is scattered across foreign servers. The patient used a number of foreign servers to upload data on. It should be noted that, the home server for one patient can be foreign for another patient.

1.4.3 Protocol Designs

Following the literature review and design requirements specification, protocols satisfying these requirements were proposed and designed. This stage of our work led to two novel contributions. The first is the design, analysis, and evaluation of novel protocols to support anonymous registration and authentication across a distributed system. The second novel contribution is the design, analysis, and evaluation of data uploading and forwarding protocols. The uploading protocol aims to allow the patient to upload data on several foreign servers. The forwarding protocol allows the patient's home server to anonymously link the data scattered across foreign servers and then forward them to the healthcare provider.

1.4.4 Analysis and Evaluation

The next stage of our research was to analyze the security and privacy properties of the designed protocols and evaluate their performance. The analysis was conducted using informal analysis verification. The informal analysis was carried out to analyze the designed protocols against the security requirements specified, in addition to the attacks and threats identified. After the completion of the security analysis and verification, the performance of the protocols was evaluated in terms of their computational and communication overheads. The SPDC protocols were compared with the most relevant work to demonstrate the effectiveness of the ideas and methods used in the protocol design.

1.5 Novel Contributions

The research work presented in this thesis has led to the following novel contributions and achievements.

- Novel Secure and ID Privacy Distributed Data Collection (SPDC) framework. This framework consists of a system architecture, methods and protocols.
  - The aim of SPDC is to facilitate collecting data from mobile patients anywhere and anytime in a scalable, secure, and ID privacy-preserving manner.
  - The SPDC architecture consists of the following: patients wearing wearable devices and carrying mobile devices, several data collection servers owned by different service providers, and the Healthcare Provider Server (HCP). Each data collection server should register with the HCP. The patient selects one server from the data collection servers to be the home server, and several data collection servers to be the patient's foreign servers. It should be noted that one patient's home server can be a foreign server for another patient.
  - The function of each server is as follows: each foreign server collects data from the patient. The home server is responsible for collecting the patient's data which is scattered across different foreign servers and forwarding it to the HCP. The HCP is the ultimate storage for patient's data, where processing and analysis takes place to make the correct decisions.
  - The purpose of using multiple data collection servers is to protect the privacy of the patient and achieve efficiency. If the patient uses one data collection server for a long time, the data collection server will be able to learn implicit attributes about the patient and therefore could identity the patient based on these attributes (i.e., communication

pattern). The reason for allowing the patient to select one server as a home is to mitigate the burden of the HCP. This is because the home server for each patient is responsible for linking the patient's data across foreign servers, instead of one entity (i.e., HCP) doing the linking job for all patients in the system.

- Novel pseudonym generation and linkage methods. These methods are built on top of the system architecture.

  - The purpose of these methods is to protect the ID privacy of the patient by providing the patient with pseudonyms instead of using his/her real ID.
  - There are two types of pseudonym: the index pseudonym and the request pseudonyms. The index pseudonym is used as an account name for the patient with the data collection server. The request pseudonym is used to uniquely identify each transaction. This is to prevent linking a number of transactions (uploads) for the same patient. Only the data collection server which the patient is using to upload can link the request pseudonyms to the patient's account (i.e., the index pseudonym).
  - The names of the pseudonyms are as follows: the home index pseudonym, the foreign index pseudonyms, the home request pseudonyms and the foreign request pseudonyms. The home index pseudonym is the patient's account ID with the home server. The foreign index pseudonyms are the patient's account IDs with the foreign servers, one for each foreign server. The home request pseudonyms are pseudonyms that are used to identify transactions with the home server, one home request pseudonym for each transaction. The foreign request pseudonyms are pseudonyms that are used to identify transactions with each foreign server, one foreign request pseudonym for each transaction (uploading).
  - There are two methods to generate and link pseudonyms: the hierarchical method and the other is the non-hierarchical method. In the hierarchical method, the pseudonyms are generated based on each other. In more detail, the home index pseudonym is generated based on the real ID of the patient. The foreign index pseudonym is generated based on the home index pseudonym. The home request pseudonyms are generated based on the home index pseudonym. The foreign request pseudonyms are generated based on respective foreign index pseudonyms. In the non-hierarchical method, the pseudonym is an elliptical curve public key.
  - For the hierarchical method, the HCP is responsible for generating and linking the home index pseudonym using HIP-Gen and HIP-Lnk algorithms respectively. The patient is responsible for generating the following: the home and foreign request pseudonyms, and the foreign index pseudonyms using HRP-Gen, FRP-Gen, and FIP-Gen algorithms respectively. The home server is responsible for linking the foreign index pseudonyms for the patient to the patient home index pseudonyms using the FIP-Lnk algorithm. This is because the home server is responsible for linking the patient's data, which is scattered across multiple foreign servers, to the patient's account in the home server. The

home server is also responsible for linking the patient's home request pseudonyms to the home index pseudonym of the patient using the HRP-Lnk algorithm. The foreign server is responsible for linking the patient's foreign request pseudonyms to the foreign index pseudonym of the patient using the FRP-Lnk algorithm. Regarding the non-hierarchical method, the home index pseudonym is an elliptical curve public key. The home request pseudonym is generated based on the home index pseudonym. The foreign index pseudonym is a temporal elliptical curve public key [3]. The foreign request pseudonym is generated based on the foreign index pseudonym.

  – The purpose of designing the non-hierarchical method is to verify the request pseudonym efficiently without searching for the shared key in the database.

- A novel Anonymous Distributed Authentication (ADA) solution.

  – The ADA solution is designed to allow only the legitimate patient to access data collection servers and use the collection service. As mentioned earlier, the data collection servers are owned by different service providers. To allow the patient to access these servers easily without the need for an identity provider to verify the patient, the ADA has two forms of credentials: digital certificates and the request pseudonyms. Digital certificates are used once to register the patient with the data collection servers. The request pseudonyms are used frequently to anonymously authenticate the patient for each request. There are two types of digital certificates the home pseudonym certificate (HCert), issued by the HCP to allow the patient to register with the home server and foreign pseudonym certificates (FCerts). The FCerts certificates are generated by the patient and blindly signed by the patient's home server.

  – ADA has Inter-Domains Anonymous Registration (IDAR) protocols and Intra-Domain Anonymous Authentication (IDAA) protocols. The IDAR allows the patient to first register with the home and foreign data collection servers using the HCert and FCerts respectively. The IDAA allows the patient to anonymously authenticate with a home or foreign server using a hierarchical or non-hierarchical authentication methods.

- Anonymous Authenticated Data Uploading and Forwarding (A2DUF) Protocols.

  – The A2DUF solution is designed to allow the patient to upload the data on any foreign servers. Then, each foreign server forwards the patient's data to his/her home server. The home server is responsible for anonymously linking the patient's data to the patient's home index pseudonym, aggregating the patient's data then forwarding it to the HCP.

  – The A2DUF uses the double signing and encryption method. The purpose of this method is to prevent any foreign server or the home server of the patient from adding fake data on behalf of the patient. This method allows the home server to verify the authenticity of the patient's data coming from a number of foreign servers. In addition, it allows the HCP to verify the authenticity of the patient's data coming from the patient's home server.

  - The A2DUF prevents any entity from inside the system or outside the system from learning patient's upload pattern, by using two key ideas: uploading on multiple foreign servers and using our invented swarming algorithm. Swarming is used to hide the pattern of uploads and make the uploading appear random. Swarming [4] is a technique used by animals (e.g. birds) to defend themselves from predators, moving in unpredictable ways to evade a predator. It is a very complex behaviour. We borrow this idea from wildlife but use it in a way that is suited to our case study.

- SPDC performance was evaluated as follows.

  - The performance evaluation of SPDC has been done using a single machine. We use a desktop computer to represent the home server, foreign server, healthcare provider server (HCP), and the mobile device of the patient.
  - To get an accurate performance measurement we use a Java benchmarking tool called Java Microbenchmark Harness (JMH). We use the JMH tool to measure the execution of each cryptographic method used in our protocol separately. The summation of the execution time for all methods used in the protocol is the overall execution time for the protocol.
  - To predict the performance metrics of our protocols, we applied queuing theory. We analyzed the performance of the protocols based on two models first when we have a single data collection server (M/M/1 single queuing model) handling patients' requests, and second when we have multiple servers (using M/M/C multiple servers model) handling the requests. Queuing theory uses mathematical formulas to theoretically analyze the performance metrics of a system.
  - The swarming algorithm is a theoretical idea and its implementation has been left for future work.

1.6 Publications

During the course of this research the following paper have been published.

- ALJOHANI, Tahani; ZHANG, Ning. A secure and privacy-preserving data collection (SPDC) framework for IoT applications. In: International Conference on Critical Information Infrastructures Security. Springer, Cham, 2020. p. 83-97.

1.7 Thesis Structure

The remainder of this thesis is organised as follows.

- **Chapter 2**: This chapter aims to find the gap in a Patient Health Monitoring (PHM) data collection system.

- **Chapter 3**: This chapter introduces the cryptographic building blocks used in the design of our methods and protocols presented in this thesis.

- **Chapter 4**: This chapter describes our innovative framework called the Secure and ID-Privacy Preserving Distributed Data Collection (SPDC) framework.

- **Chapter 5**: This chapter describes two methods that are used to generate and link pseudonyms that the patient uses with the servers to hide their real ID and to prevent linking multiple uploads made by the same patient with one server. This chapter describes hierarchical pseudonym generation and linkage methods and non-hierarchical pseudonym generation and linkage methods. The non-hierarchical method is designed to enhance the pseudonym verification time without searching for the verification key in the server's (verifier) database.

- **Chapter 6**: This chapter presents the design and evaluation of novel secure protocols for distributed authentication, known as Anonymous Distributed Authentication (ADA) protocols. The ADA has two protocols, one for registration across multiple domains and the other for session authentication in one domain (a data collection server).

- **Chapter 7**: This chapter describes an Authenticated Data Uploading and Forwarding (A2DUF) Protocol Suite to support uploading data on any foreign servers. This data is linked by the patient's home server which forwards the data after aggregation to the healthcare provider.

- **Chapter 8**: This chapter concludes this thesis and suggests directions for future research.

Chapter 2

Patient Health Monitoring (PHM) Systems: Problem Identification

2.1 Chapter Introduction

This chapter analyses data collection systems used for Patient Health Monitoring (PHM). The aim is to identify knowledge gaps in existing PHM data collection systems. This chapter starts with a description of various PHM systems and the most important component in these systems which is the data collection system. It then describes a typical use case scenario and derives a generic model of the data collection system. Then, threat analysis is conducted based on the generic model. As a result of the threat analysis using the model, the chapter specifies a set of requirements for the design of a secure and efficient PHM data collection system. Against these requirements, the chapter critically analyzes existing data collection systems and solutions published in the literature, highlighting knowledge gaps and our ideas to address them.

In detail, the chapter is organized as follows. Section 2.2.1 explains what PHM is and presents classical PHM systems. Section 2.3 specifies the design requirements. Section 2.4 critically analyses existing data collection systems against the requirements. Section 2.5 highlights knowledge gaps, or areas for improvement, in existing solutions. Section 2.6 presents our ideas to address the knowledge gaps. Finally, Section 2.7 provides a summary of the chapter.

2.2 PHM System Overview

2.2.1 What is a PHM System

A PHM system can be defined as the use of computing and wireless communication technologies to provide periodic, on-demand, and on-event data collection from a patient anywhere and anytime. It then electronically transmits the collected data to a health care provider (e.g. hospital) for assessment and provides recommendations for the remote patient [5]. A typical PHM system [6]–[8], works as follows. First, the wearable sensors read the health data (e.g.

heart rate) from the patient's body. They then aggregate and send the health data to the gateway device via a wireless connection (e.g. Bluetooth). Second, the gateway device collects the health data from wearable sensors. The gateway device performs basic processing and analysis of the data to extract features and detect any abnormality in the patient's health status. Third, the gateway device delivers the health data to the healthcare provider. The deliveries are at different frequencies. These are periodic, on-demand, and on-event deliveries. The periodic and on-demand deliveries mean that data is delivered to the healthcare provider at regular intervals (e.g. every hour), and upon request. The on-event frequency means that data is delivered to the healthcare provider upon detection of an abnormality in the patient's health status (e.g. breathing problems). The healthcare provider receives the health data, analyses the data to make proper decisions then stores the patient's data.

2.2.2 Types of PHM Systems

Based on our literature review, we can divide the PHM systems into three categories: first generation systems, second generation systems, and third generation systems. The first generation systems use a fixed gateway to collect the patient's data. This restricts the patient's mobility as they have to be within a specified vicinity (i.e., within a hospital or a patient's home). Examples of this generation of system can be found in [9]–[12]. The second generation systems use a mobile gateway (e.g. mobile phone). This allows free mobility to the patient. Both the first and second generations are built based on on-premises infrastructure. With on-premises infrastructure, all health monitoring services are managed and maintained within the healthcare provider on their hardware and servers. The problem with on-premises infrastructure is that it lacks scalability when the number of patients increases. Examples of this type of system can be found in [13]–[15]. The third generation systems solve the scalability problem by using a cloud infrastructure where a third-party provider hosts the hardware and software for the healthcare provider. Examples of this type of system can be found in [16]–[20].

2.2.3 Data Collection in PHM Systems

The operation of a PHM system typically has three stages: data collection, data analysis (i.e. the analysis of the collected data), and decision making (based on the outcome of the data analysis). The data collection stage is crucial to the correct running of a PHM system, as the correctness of the analysis and decision making depend on the correctness of the data collected. The correctness of the data collected requires that the right data is collected at the right time and its authenticity is not compromised from its collection. In addition, some of the data may require confidentiality protection for reasons such as preserving data owners' privacy. Therefore, a secure and efficient data collection system is essential for successful delivery of quality PHM services to patients. Therefore, our focus in this thesis is to design a secure, ID privacy-preserving, and efficient data collection system to support data collection from mobile patients. To achieve this aim, we first specify the design requirements.

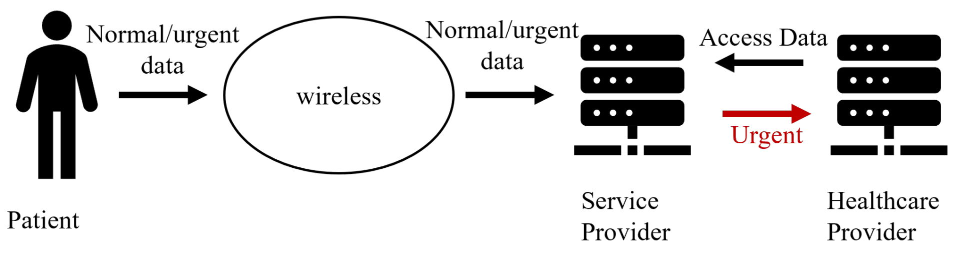


Figure 2.1: Generic model

2.3 Specification of Requirements

In this section, we start by analyzing a use case scenario driven from our investigation of the third generation of PHM systems. Subsequently, based on the analysis of the use case scenario, we construct a generic use case model to study security and performance concerns. This then leads to the specification of a set of design requirements for the design of a secure, ID privacy-preserving, and efficient data collection system.

2.3.1 A Use Case Scenario

According to Microsoft [1], the UK government has taken an important step in providing clarity to the National Health Service (NHS) in storing patient information in the cloud and providing health service. We use the following scenario to highlight the potential threats of using a third service provider to collect patients' data.

A patient, Alice, suffers from a heart disease that needs to be monitored very frequently. Alice's healthcare provider (hospital) conducts monitoring via the use of wearable devices. These devices collect Alice's health data in real-time and send the data to a service provider (e.g. Microsoft) via her mobile device. This data collection process is performed regularly, say every 5 minutes. If the collected data indicates that Alice may have a health problem, the service provider notifies the healthcare provider, which may take further action. Alice's healthcare provider can access the data collection service hosted by the service provider.

2.3.2 A Generic Use Case Model

Based on the use case scenario above, a generic use case model of the PHM data collection system is constructed, as shown in Figure 2.1. The model includes the following entities.

- Wearable sensors: These sensors are worn by a patient and responsible for acquiring health data from the patient's body and transmitting the data to the patient's mobile device.

- A mobile device: This is a mobile device carried by the patient. The mobile device runs a health application to collect health data from wearable sensors, and processes and transmits it to the service provider.

- A service provider: This is a third-party provider (e.g. Microsoft) responsible for collecting and storing the patient's data. If the provider receives an urgent message from the patient, the provider forwards the message directly to the healthcare provider.
- Healthcare provider: This is the hospital that the patient has registered with. The hospital can access its patients' data collected by the service provider for further analysis.

2.3.3 Threat Analysis

There are a number of security and privacy concerns or threats in the above generic model. They are listed below. Here, the service provider is authorized to collect the data, but not authorized to gain access to the data, nor the real identity (ID) of the owner of the data (i.e. the patient). They should not know anything about the patient's real identity and the health data.

- Authentication threats: An unauthorized entity may try to impersonate a patient to gain unauthorised access to the patient's data stored on the service provider. Such impersonation attacks are sometimes referred to as "identity theft" in the literature.
- Data authenticity threats: The patient's data may be delayed, replayed, or even modified. An unauthorized entity may try to forge data to make it seems as though it is from a legitimate patient. Such attacks, if successful, may lead to a break in the data collection system or compromise the integrity of data stored in the system, leading to serious consequences for patients using the system.
- Data confidentiality threats: If the patient's data is not protected during transit or in store, and if an unauthorized entity gains access to the patient data, e.g. by eavesdropping the channel, the patient's private and sensitive information, such as the patient's identity and medical conditions maybe exposed to unauthorized entities. The exposure of patients' private or sensitive information could have serious consequences for them. For example, some insurance companies may avoid insuring some patients.
- Patient's ID privacy threats: If the patient is using different artificial IDs when communicating with the same service provider, a threat known as an "inference threat" could occur. This threat results from using the same service provider over a long period in which the service provider can use the pattern of communication to infer information (e.g. IDs) about the patient.

2.3.4 Design Requirements

To design a data collection system that could accomplish the task of collecting health data from remote patients anywhere anytime in a secure, privacy-preserving, and efficient manner, we specify a set of requirements. The requirements can be classified into functional (F), ID privacy-preservation (P) and security (S). The details of these requirements are as follows.

(F). The system should support various modes of data collection: The collection of data should be such that various modes of collection are supported. These are periodic and event-driven data collection modes. In addition, the data collection system should support patient mobility everywhere (i.e. it should not be restricted to a home or hospital).

(P). The system should preserve patient's ID privacy: To satisfy this requirement, the following three requirements (P1,P2,P3) should be satisfied.

> (P1). The system should preserve patient's real ID anonymity: The real ID of the patient should be anonymous. Only the healthcare provider can learn the real ID of the patient.
>
> (P2). The system should support unlinkability: Different uploads (one session may have a number of uploads) by the same patient should be unlinkable to any unauthorized entity.
>
> (P3). The system should prevent attribute disclosure: The uploading pattern of the patient should be hidden.

(S). The system should ensure entity authentication and data confidentiality: To provide these assurances, the following three security requirements (S1,S2,S3) should be satisfied.

> (S1). The system should support mutual entity authentication: Entity authentication ensures that a communicating entity is indeed who it claims to be. This requirement is to counter impersonation attacks. This requirement should be satisfied without compromising the patient's ID privacy.
>
> (S2). The system should support end-to-end data authenticity: Data authenticity assures that data are indeed from the claimed source and that it is the same as has been sent by the original sender. This requirement is to counter tampering, replay, or forgery attacks on data in transit.
>
> (S3). The system should support end-to-end data confidentiality: Confidentiality protects data against unauthorized disclosure. This requirement is to protect against unauthorized access to data in transit.

(E). The design should be as efficient and as scalable as possible: The data collection system should be scalable in that, as the number of patients and/or the data generated by patients increases, the overhead costs imposed on the healthcare provider should not increase sharply.

2.4 Existing Data Collection Systems

In this section, we examine data collection systems based on the specified design requirements. This is to identify a knowledge gap. The data collection systems can be grouped into two categories: (1) centralized systems, and (2) distributed systems.

2.4.1 Centralized Systems

A centralized data collection system only uses a single central data server to which patients' data are uploaded.

In [21] the authors proposed a federated cloud and Internet of Things (IoT) health monitoring system. The system consists of patients wearing wearable sensors, patients' mobile devices, a data centre, and a healthcare provider. The sensors periodically measure the health status of the patient, collect, and then send the collected data to the patient's mobile device. The mobile device pushes the data to the data centre where the data is analyzed and stored. The results from the analysis are sent to the healthcare provider. This system is considered scalable as it relies on cloud infrastructure and technologies (such as virtualization) to achieve scalability. However, the system fails to satisfy a number of security and privacy requirements, such as ensuring patients' data confidentiality and authenticity, and patients' ID privacy.

Similar to [21], authors in [22] proposed a secure fog-based smart health service. The patient data is collected by heterogeneous fog layers. Each fog layer consists of distributed small-scale network resources (e.g. gateways, cloudlet servers, and routers). The health data collected by fog nodes is sent to the healthcare provider for further analysis. This system supports data encryption and confidentiality. The requirement for preserving patients' ID privacy is not considered.

In [23], the authors proposed a privacy-preserving priority classification scheme, called PPC. This scheme enables only authenticated patients to report their encrypted health data periodically to a gateway. The gateway receives the encrypted data, classifies it, and reports it to the healthcare provider server according to their priority. The architecture of the system consists of wearable sensors, a mobile device, a number of gateways (deployed in the patients' vicinity), and a healthcare centre. The wearable sensors collect the health data and send it to the mobile device, which encrypts the health data and calculates the threshold of the data (i.e. how critical the data is). The device then sends the encrypted health data to the gateway. The gateway receives the encrypted health packet and classifies the health packets according to a threshold number. The proposed system does not consider the ID privacy of the patient.

The authors in [24] present a privacy-preserving aware data transmission for IoT-based health applications. The proposed system preserves the privacy of a patient's data and the contextual data. To preserve the patient's data privacy, the patient's health data is encrypted. To preserve contextual privacy the proposed system uses a building path algorithm and broadcasting strategy. The encrypting method uses a novel lightweight identity-based encryption (L-IBE) that is built based on an elliptical curve discrete logarithmic problem. The privacy part of this scheme is built on the top of the L-IBE, which includes a novel building path algorithm called BPA. To ensure contextual privacy, the scheme uses the following algorithms. The building path algorithm uses injecting fake data and multi-casting strategy. This algorithm allows the patient to send encrypted messages to a patient in the neighborhood, who in turn may send the message to another patient or send it directly to the IMSS. Although this work satisfies a number of security and privacy requirements, it relies on a single entity (i.e. IMSS) to collect and store the health data of the patient. Over time, the entity could learn sensitive attributes for each patient. These attributes could be used to re-identify the patient.

The authors of [25] proposed a secure and privacy-preserving opportunistic computing (SPOC) framework for a mobile healthcare emergency. Opportunistic computing can be defined as exploiting available computing resources near the patient to execute the task and report the health data of the patient as quickly as possible. In other words, an emergency patient may not have enough computing and storage resources in his/her mobile device to report the health data to the healthcare provider. In this case, the patient can request the near patient executes the task on his/her mobile device. The main challenge with the opportunistic computing approach is the disclosure of the patient's health data. To solve this issue, SPOC provides an efficient attribute-based access control mechanism. This mechanism allows the patient in an emergency case to identify other nearby patients who have similar symptoms without directly revealing the patient's symptoms to other patients. However, the existence of mobile health social networks is necessary to use the solution provided by SPOC.

Similar to [25], the authors in [26] proposed PEC, which stands for Privacy-Preserving decentralized Emergency Call scheme for mobile healthcare social networks. It allows the patient in a life-threatening situation to quickly and accurately transmit emergency data to a nearby physician for help or forward the call to a nearby patient who in turn forwards the call to a physician. PEC is a mechanism to protect the confidentiality of the patient's data so only authorized users can decrypt it. This is done by using an access control mechanism that allows only authorized users to learn the secret key which is used to encrypt and decrypt the health data. PEC also protects the real ID of the patient as the health message is signed using the group signature.

The authors in [27], proposed SAGE which stands for Strong privacy-preserving scheme Against Global Eavesdropping for e-health system. SAGE protects the confidentiality of the health data and cuts off the relationship between the patient and his/her physician. This is because if an observer knows that the patient often sends the data to a particular physician, the observer may use this information to infer the disease which the patient has. SAGE preserves the content and contextual privacy. This is done by first encrypting the health data and then sending the encrypted data to a health centre. The centre is responsible for broadcasting the data to all physicians. Then, only the potential physicians will be aware of the data of their patients.

2.4.2 Distributed Systems

The architecture of a distributed data collection system has many servers. These servers are geographically distributed to collect health data. Unfortunately, many of these systems only consider the scalability of the system architecture but neglect numerous security and privacy issues. Next, we review some of these distributed data collection systems.

In [28], the authors proposed an e-health monitoring system based on geographically distributed clouds. The distributed clouds consist of many cloud servers deployed over a large region. When a user wants to upload health data to the system, the geographically closer server to the user handles the request and checks the workloads of other servers. Then, it assigns a server to the user to upload the data. After receiving the response, the user applies a traffic-shaping algorithm on his/her health data before uploading the data to the assigned server. The traffic-shaping algorithm preserves the privacy of the patient by hiding the original sender of the health data. Similar to this work, the author in [29] proposed distributed cloudlets which are scattered across a geographical area (e.g. a city). In [29], the author mentioned the importance of the security of the data but has not considered any mechanism to fulfill this aspect.

In [30], the authors proposed remote monitoring based on blockchain technology called Healthchain. In Healthchain, there are a number of patients and doctors, blockchain nodes, and distributed file system (DFS) servers. The patient first encrypts the health data and then uploads it to a DFS server. Only the hash string of the encrypted health data is published as a data transaction in the blockchain. The transaction contains the patient's ID, the time, the hash string, and the patient's signature using the patient's private key. After this, the patient publishes a key transaction that contains the patient's ID, the key that the patient used to encrypt the health data, and the doctor's ID. The doctor retrieves both the hash string of the patient data and the secret key from the blockchain using the patient's ID. They then use the hash string to retrieve the patient's encrypted data from the DFS servers. The doctor then decrypts and analyzes the health data. The architecture of the system does not rely on a single central point but it supports large-scale health data and has good scalability. In Healthchain, the authors neglect the patient's ID privacy.

Other distributed data collection systems are designed to collect non-health data. We will discuss these next. The authors in [31] proposed a novel system for a secure and privacy-preserving cloud video reporting service in vehicular networks. The proposed system allows users to report videos of accidents to designated vehicles such as an ambulance. This aims to provide a timely response toward traffic accidents, which enhances road safety. The proposed system consists of a number of vehicles, cloud servers, trusted authorities (TA), and a department of motor vehicles (DMV). The system works as follows. When a vehicle's camera captures a video of an accident, it performs the following. First, it encrypts the content of the video using a symmetrical key and cipher-text-policy attribute encryption algorithm. Second, it selects a new pseudonym ID (several pseudonym certificates were issued by TA). Third, it constructs a message to be sent to a designated vehicle. The recipient decrypts the content only if it meets the encryption policy. The proposed system satisfies the security and privacy requirements of authentication, and non-repudiation using a public key-based digital signature. In addition, it satisfies confidentiality by using encryption and anonymity by using pseudonym authentication.

In [32], the authors proposed a system which allows a vehicle to request the optimal route from its starting point to a final destination. The vehicle first submits a navigation query to the

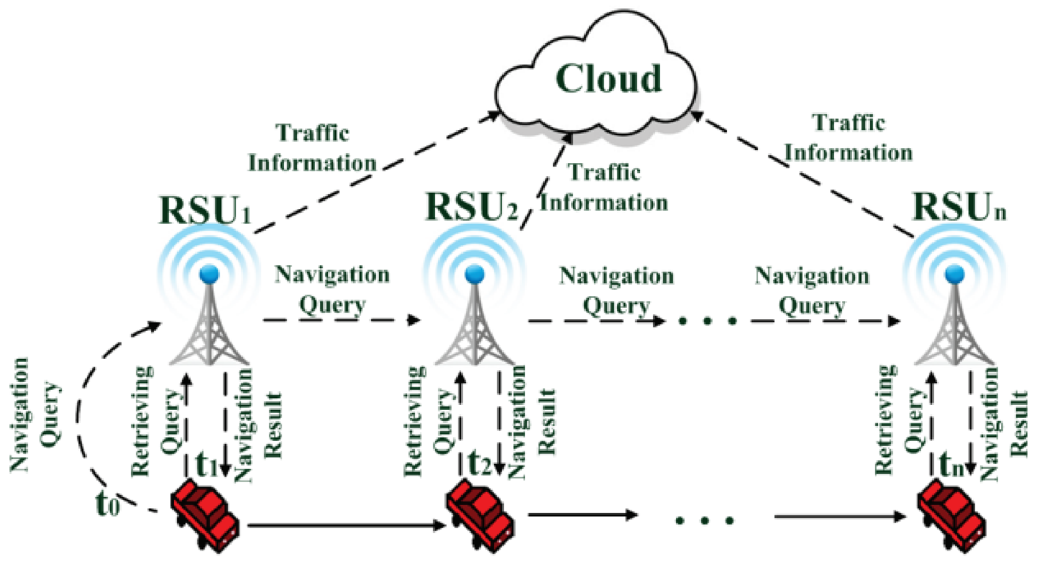


Figure 2.2: The system architecture of [32]

Road Side Unit (RSU). Each RSU receiving the query performs the following steps: i) sends the query to the next RSU, and ii) sends a crowd-sourcing task to the vehicles in its region. The queried vehicle can receive an updated navigation result for the optimal path route from each RSU. Each vehicle can hide its real identity when it requests a navigation query from RSU or when it sends a result of a crowd-sourcing task to RSU. This is done by using an encryption mechanism and anonymous credentials. Figure 2.2 shows the system architecture.

In a disaster area, the connectivity of a remote health monitoring system is not reliable because of the damage that occurs in the communication infrastructure. Therefore, other researchers [33] suggest integrating the health monitoring with other infrastructure such as vehicular systems. This is to facilitate reporting health data by patients in the disaster area as quickly as possible. The authors presented mvSERS [33], which is a secure real-time monitoring emergency response system. The system architecture of mvSERS consists of: wearable sensors, smartphones, in-vehicle response, and medical emergency response systems (EMC). The wearable sensors send the health data to the smartphone which measures any abnormalities in the health data. If it is high then it encrypts health data, generates a signature and then sends it to the EMC. Otherwise, it sends the health data to the in-vehicle response system. The mvSERS satisfies basic security requirements such as data confidentiality by encrypting the health data, and providing data authenticity and integrity using a digital signature.

From the above discussions, we can see that existing centralised data collection systems offer better support in terms of ensuring patients' data security and privacy than their distributed counterparts. However, these systems are not scalable. The existing distributed data collection systems, on the other hand, offer better scalability, but they have not addressed the security and privacy issues properly. The analysis of the related work against the requirements is summarized in Table 2.1, where F means the system provides functional requirements, P1 means the system provides anonymity, P2 means the system support unlinkability, P3 means the system prevents attribute disclosure, S1 means the system provides mutual entity authentication, S2 means the system provides end-to-end data authenticity, S3 means the system

Related Works	F	S1	S2	S3	P1	P2	P3	E
[21]	p	✗	✗	✗	✗	✗	✗	✗
[22]	p	✗	✗	✓	✗	✗	✗	p
[28]	p	✗	✗	✗	✓	✗	✗	✓
[29]	p	✗	✗	✗	✗	✗	✗	✓
[31]	✓	✓	✓	✓	✗	✗	✗	✗
[32]	p	✓	✓	✓	✓	✓	✗	✗
[23]	✓	✓	✓	✓	✗	✗	✗	p
[24]	✓	✓	✓	✓	✓	✓	✗	p
[25]	p	✓	✓	✓	p	✗	✗	✗
[26]	✓	✓	✓	✓	✗	✗	✗	p
[27]	p	✓	✓	✓	✓	✗	✗	p
[33]	✓	✓	✓	✓	✗	✗	✗	p

Table 2.1: Analysis of Related Works

provides end-to-end data confidentiality, E means the system is scalable and p in Table 2.1 means the requirement is partially satisfied.

2.5 What is Missing?

Findings from our critical analysis of existing data collection systems in the section above can be summarized as follows:

- None of the solutions cover all the specified requirements in Section 2.3.4. Some solutions satisfy a number of security and ID privacy requirements but their architecture is not scalable (i.e. centralized). Other solutions have a distributed architecture but neglect a number of security and privacy requirements. A few solutions satisfy scalable architecture, security and privacy but restrict the patient's mobility.
- None of the existing solutions have considered inference attacks, by which an unauthorized entity may be able to infer a patient's ID by observing the communication patterns between the patient and the service provider.

2.6 High-level Ideas

We aim to build a competitive data collection system as data collection is a crucial stage in a PHM system. To achieve this aim, we needed to know the design requirements for the data collection system. To learn the requirements, we analyzed a use case scenario. After the analysis, we came up with functional, security, ID privacy, and efficiency requirements specified in the Section 2.3.4. We concluded that our competitive data collection system should satisfy all requirements and address the security threats identified in Section 2.3.3. This section presents some recommendations to achieve a secure, ID privacy, and efficient data collection system.

- Design a new architecture for the PHM data collection system: The typical data collection system is presented in Figure 2.1. It relies on one service provider (e.g. Microsoft) to collect the health data from different patients. The main problem with this model is that it does not protect against inference attacks. For example, Alice uploads some data (such as blood pressure) at 1 pm every day and other data (such as heart rhythm) every 10 minutes. Then the person managing the service may identify Alice by discovering the pattern of uploads (i.e., 1 pm and every 10 minutes), even if Alice uses different pseudonyms to access the service. To address this problem we recommend that different service providers (or different servers managed in different domains) are used to collect the health data from the patient. For example, rather than relying on Microsoft exclusively to collect patients' data, the NHS can use multiple data collection services provided by different providers e.g. Amazon, Google, Barclays). These providers only collect the data and forward the collected data to the NHS data centre. Furthermore, the patient should select a different service provider each time. For a higher level of privacy preservation, such selections should not exhibit any pattern. We hypothesize that, in this way, the patients' ID privacy can be protected against inference attacks. This recommendation is to support the ID privacy requirement (P3).

- Use anonymous credentials to access different service providers: This is to support the authentication requirements (S1) and ID privacy requirements (P1 and P2). To ensure authorized use of the data collection service, a patient should be identified and authenticated before they are allowed to access the servers and this should be achieved without compromising the patient's ID privacy. This means that the patient identification and authentication should be carried out anonymously.

- Use multiple data collection modes and a differential data delivery approach: Patients' data is classified into two categories: normal data and urgent data. Normal data should be aggregated by data collection servers before being forwarded to the final destination (i.e., the healthcare provider server). Urgent data should be forwarded directly to the healthcare provider server without further processing. This is to reduce any intermediate delays so that the data can reach the destination as soon as possible. The data collection modes that our data collection system support are periodic and on-event data collection modes.

- Use data encryption and signing: This is to support the security requirements (S2-S3). Before a patient uploads data to any of the service providers, they should first sign the data using his/her secret key and then encrypt the data and the signature using the same key. The key is only known to the patient and the healthcare provider. The signature ensures authenticity, while the encryption ensures confidentiality.

2.7 Chapter Summary

This chapter has introduced and reviewed different generations of PHM systems, with a focus on the data collection stage. It has then discussed security, ID privacy, and efficiency issues in a PHM data collection system, and specified a set of functional, secure and ID privacy requirements for designing a secure and efficient PHM data collection system. Based on this set of requirements, the chapter has critically analyzed existing data collection systems, both in the health and non-health domains to identify knowledge gaps. Finally, the chapter has proposed new ideas that are used to address the gaps. The next chapter presents the building blocks to be used to implement these ideas, leading to the design of our secure, ID privacy-preserving, and efficient data collection system.

Chapter 3

Cryptographic Building Blocks

3.1 Chapter Introduction

This chapter introduces the cryptographic building blocks used in the design of our methods and protocols presented in Chapters 5, 6, and 7. In detail, Section 3.2 presents the public key cryptosystem, which includes RSA and Elliptic curve cryptography (ECC), Section 3.3 presents symmetrical key cryptosystem, Section 3.4 presents Hashed Message Authentication Codes (HMACs), Section 3.5 presents digital signature, which includes blind signature and proxy blind signature, Section 3.6 presents digital certificate, and finally Section 3.7 concludes the chapter.

3.2 Public Key Cryptosystem

Public key cryptography [34], as seen in Figure 3.1, is a system that uses key pair of a public key and a private key. The public key is used for encryption, while the private key is used for decryption. For example, a sender uses the receiver's public key to encrypt the message, the receiver uses its private key to decrypt the message. There are two applications of public key cryptography. First, are encryption and decryption mechanisms. For the encryption, a sender needs to obtain the public key of the receiver, then encrypt the message using the receiver's public key. The receiver can decrypt the message by using its private key. The encryption process guarantees the confidentiality of the message. The second use is a digital signature, where the sender signs a message with the private key to create a digital signature over the message. The message and the digital signature are both received by the receiver. The receiver can use the sender's public key to verify the digital signature. The digital signature guarantees the integrity and the authenticity of the message, and the sender cannot deny sending the message (non-repudiation). Public key cryptography is also known as asymmetrical cryptography.

3.2.1 Rivest–Shamir–Adleman (RSA)

RSA (Rivest–Shamir–Adleman) is one of the first public key cryptosystems [34]. RSA was invented by Ron Rivest, Adi Shamir, and Leonard Adleman. The security in RSA surrounds

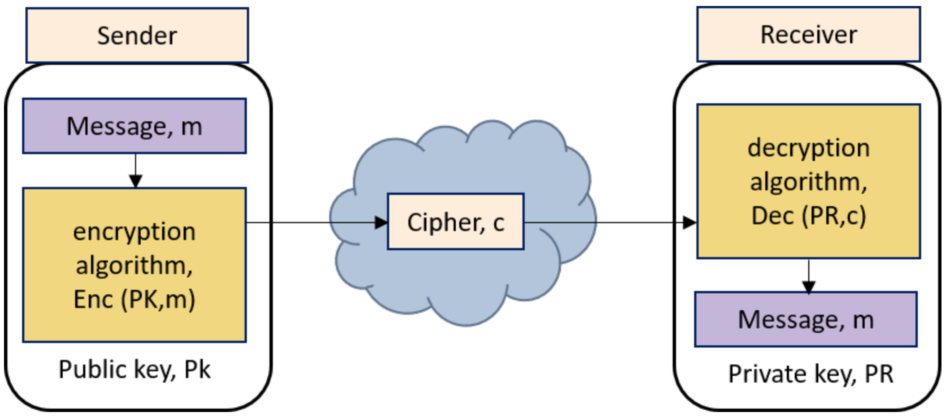


Figure 3.1: Asymmetrical key cryptosystem.

solving the prime factorization problem. This means that one can multiply two large prime numbers. The result of the multiplication is difficult to be factorized. For example, suppose there are two prime factors 1223 * 1987 = 2430101, if one learns the number 2430101, it is very hard to guess its factors (i.e., 1223 and 1987). The RSA comprises three algorithms: a key generation algorithm (KeyGen), an encryption algorithm (Enc), and a decryption algorithm (Dec). In the following algorithms Alice is a user.

- KeyGen:
  - Alice generates the RSA modulus: The initial procedure begins with the selection of two prime numbers namely p and q, and then their product is calculated n such as follows $n = p * q$.
  - Alice obtains the number (e): The number e should be greater than 1 and less than (p-1) and (q-1). The condition is that there should be no common factor of (p-1) and (q-1) except 1.
  - Alice forms the public key : The pair of numbers n and e forms the RSA public key.
  - Alice calculates the private key: Private Key d is calculated from the numbers p, q, and e as follows, $e * d = 1 \mod (p-1)(q-1)$.
- Enc: Consider Alice who sends a text message (text) to Bob whose public key is (n,e). To encrypt the text message, the following syntax is used,
  $Cipher = text * e \mod n$.
- Dec: Consider a receiver, Bob, has the private key (d), the decryption process is calculated as $Plain = Cipher * d \mod n$.

3.2.2 Elliptic Curve Cryptography (ECC)

The ECC is a member of the public key cryptosystem family [3]. The ECC was invented by Neal Koblitz and Victor S. Mille in 1980. The ECC consists of the following protocols:

key generation, key exchange and agreement, and digital signatures. In the key generation protocol, public and private keys are generated. The private key is used to sign a message and the public key is used to verify the signature on the message. In the key exchange and agreement protocol, two entities exchange their public keys and each entity multiplies the other entity's public key by its private key to generate a shared key. The ECC can perform encryption and decryption operations by using a symmetrical encryption scheme (e.g. AES), in which the shared key that is generated from the key exchange and the agreement protocol is used as an input to the symmetrical encryption algorithm. The ECC's security lies in solving the elliptic curve discrete logarithm problem (ECDLP). This problem has been shown to be computationally infeasible over the years. The ECC has the same level of security as provided by the RSA with a smaller key size. The same level of security we would expect from the the RSA 1024-bit key size is provided by a 160-bit key size from the ECC. With a smaller key size we require less memory, bandwidth, and have more efficient computations. These features of small key size, efficient computation, and smaller memory requirements, make the ECC a better choice for resource-constrained devices such as mobile devices. The following defines the elliptical curve field and equation.

The elliptic curve over a finite field : $\mathbb{Z}_p$, p > 3, is the set of all pairs (x,y) in $\mathbb{Z}_p$ which satisfies the equation $E : y^2 \equiv x^3 + ax + b \pmod{n}$ where $a, b \in \mathbb{Z}_p$, and the following condition should be satisfied $4{\cdot}a^3 + 27{\cdot}b^2 \neq 0 \pmod{n}$.

The domain parameters which define the elliptic curve are (t,a,b,$\mathbb{P}$,n), where n is the module prime, a and b are coefficients of the elliptic curve equation, $\mathbb{P}$ is the generator point, and t is the number of points in the field.

The ECC comprises four algorithms: a key generation algorithm (EKeyGen), a key exchange and agreement algorithm (EKeyExg), a signature generation algorithm (ESigGen) and a signature verification algorithm (ESigVer). We assume two entities A and B execute the following algorithms:

- EKeyGen:
  - A chooses a random private key (EPK), where $0 <$ EPR $< t$
  - Computes a public key, EPK as EPK= EPR*$\mathbb{P}$, where the operation (*) means multiplication.

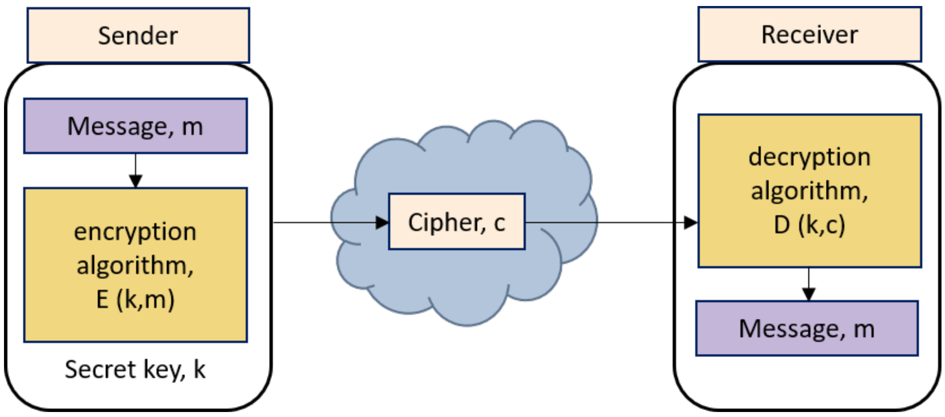


Figure 3.2: Symmetrical key cryptosystem.

- EKeyExg:
  - Both entities A and B exchange their public keys, $\mathrm{EPK_A}$, $\mathrm{EPK_B}$
  - A and B compute their shared secret key (S) as follows, $\mathrm{S} = \mathrm{EPR_A} * (\mathrm{EPK_B}) = \mathrm{EPR_B} * (\mathrm{EPK_A})$.
- ESigGen:
  - A selects an integer as a random private key ($\mathrm{EPR_r}$), where $0 < \mathrm{EPR_r} < t$
  - A computes $\mathrm{R} = \mathrm{EPR_r} * \mathbb{P} = (x_R, y_R)$, Let r = $\mathrm{x_R}$.
  - A computes a signature on a message (x) as $s \equiv (h(x) + \mathrm{EPR} * \mathrm{r}) * \mathrm{EPR_r^{-1}} \pmod{n}$. The h(x) is a hash function.
- ESigVer
  - A verifier computes an auxiliary value w, $w \equiv s^{-1} \pmod{n}$.
  - The verifier computes an auxiliary value $u1$, $u1 \equiv w * h(x) \pmod{n}$.
  - The verifier computes auxiliary value $u2$, $u2{\equiv}w * r \pmod{n}$.
  - The verifier computes $Q = u1 * \mathbb{P} + u2 * \mathrm{EPK_B}$. This results in a point (x_Q, y_Q).
  - The verifier checks if $r = x_Q \pmod{n}$ signature is valid.

3.3 Symmetrical Key Cryptosystem

A symmetrical key cryptosystem, as seen in Figure 3.2, uses the same secret key by a sender and a receiver to perform both encryption (E) and decryption (D) operations. One example of the symmetrical key cryptosystem is an Advanced Encryption Standard (AES) [34]. The symmetrical key cryptosystem is used to provide message confidentiality.

- E: The encryption operation can be performed as follows. The sender uses a message (m), and a key (k), as inputs to an encryption algorithm (E). This results in generation of an encrypted message (c). This process is denoted as, $c = E(k, m)$.

- D : The encrypted message is then sent to the receiver through a communication channel. Once it is received, the receiver uses the ciphertext as well as the same key as inputs to a decryption algorithm, D, which recovers the message. This process is denoted as, $m = D(k, c)$.

3.4 Hashed Message Authentication Code (HMAC)

A Hashed Message Authentication Code (HMAC) [34], as seen in Figure 3.3, integrates the uses of symmetrical encryption with a hash function. The hash function is any function that can be used to take a message of any length (m) and convert it to fixed-size values (h). The hash function has the following properties.

- One-way function: Given a hash value h it should be difficult to find any message m such that $h = hash(m)$.

- Collision resistance: It should be difficult to find two different messages $m1$ and $m2$ such that $hash(m1)$ = $hash(m2)$.

The HMACs operation involves using a secret key known to both the sender and the receiver. The HMAC can be explained as follows. First, both a sender and a receiver agree upon a secret key. Then, the sender uses the secret key to generate the HMAC on a message, m, i.e. $h = H(k, m)$. Then the sender sends the message, m, and the generated HMAC, h, via a secure message to the receiver. Once it is received, the receiver uses the same message, m, and the secret key, k, to generate the HMAC, i.e. $h = H(k, m)$. The hash value generated by the receiver is then compared with the one received from the sender. If the values of both HMACs are the same, we can say that the message is an authentic one. This means that the message is from the claimed sender and the message has not been modified since the time was sent. The HMAC is used to prove the authenticity of the message.

3.5 Digital Signature Schemes

3.5.1 Proxy Signature

In 1996, Mambo et al. [35] proposed the concept of a proxy signature. To understand the proxy signature one can use the following analogy. In a business environment, if a manager delegates the signature to a subordinate, the delegation process allows the subordinate to sign the documents on behalf of the manager. However, in a digital business environment, the delegation process (i.e. signing documents digitally) becomes a challenge. To sign a document

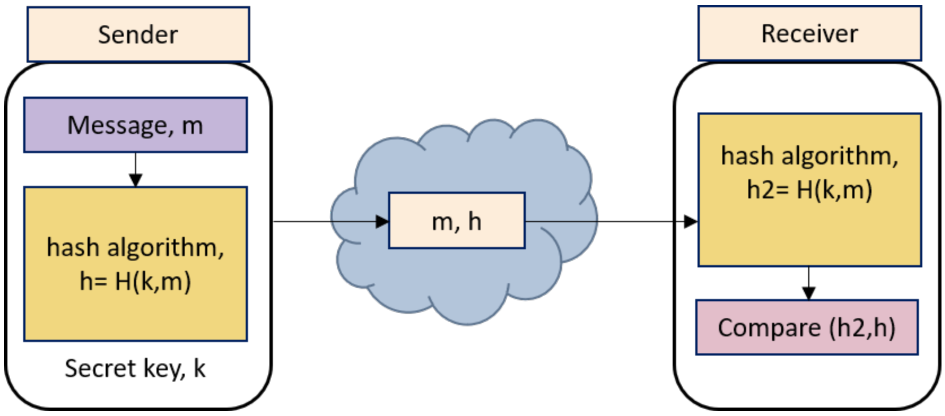


Figure 3.3: Hashed message authentication code.

digitally on a behalf of the manager, one can use the proxy signature scheme. In this scheme, there is one original signer (the manager) and one or more proxy signers (subordinates). In a proxy signature scheme the original signer delegates their signing capability to the proxy signer.

To ensure that only an authorized proxy signer generates a signature, we use the warrant-based proxy signature. This idea can be explained as follows. The original signer generates proxy public and private keys, then generates the signature using the proxy private key on the warrant. The warrant contains important information. Subsequently, it sends the warrant to the proxy signer. The proxy signer generates their public and private proxy keys and generates the private proxy key based on the warrant [36].

3.5.2 Blind Signature

A blind signature is a kind of digital signature in which a signer signs a message without knowing the content of the message, but a third party can later verify the signature and know that the signature is valid. The blind signature is made by, first, the message being blinded by a requester using blind factors. Then the requester sends the blind message to the signer. Second, the signer signs the message using its proxy private key. Third, after receiving the signed message from the signer, the requester can derive the valid signature from the message by deducing the blinding factors. Any verifier after that can check the validity of the blind signature using the public proxy key of the signer. Figure 3.4 illustrates the concept of blind signatures [37]. Our proxy blind signature scheme is based on elliptic curve cryptography (ECC) [38].

3.5.3 Proxy Blind Signature

This method is executed by the original signer to delegate the signing capability to each proxy signer. The delegation method works by the original signer generating a warrant (m_w) which contains important information such as the validity period of the delegation, the identities of both the original signer and the proxy signer. The purpose of the warrant is to explicitly state

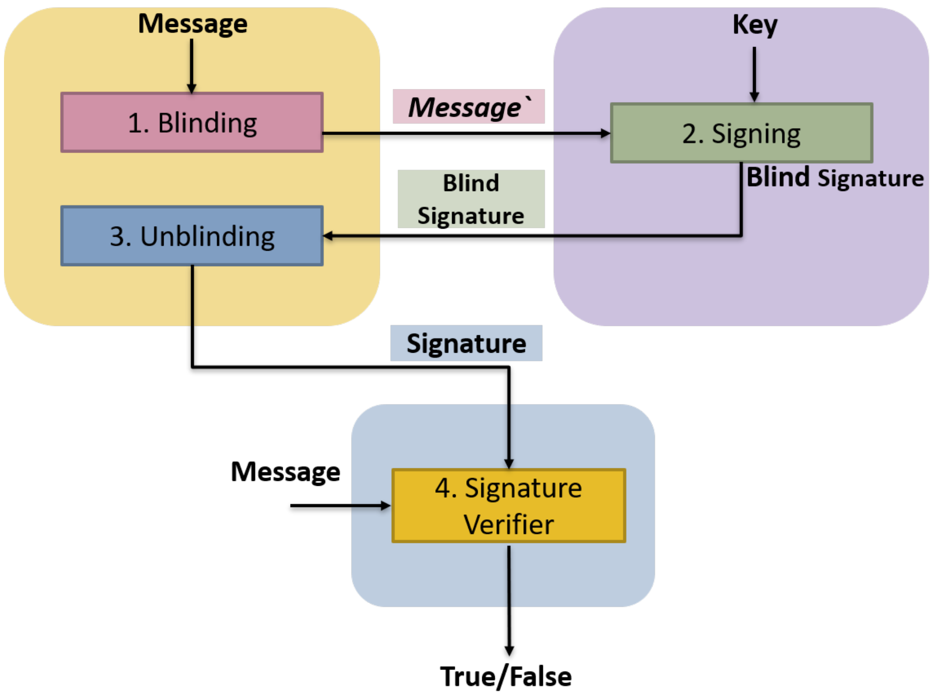


Figure 3.4: Blind signature mechanism.

that the proxy signer is a certified proxy signer by the original signer and to prevent the proxy signer from delegating the power of signing to another party. The proxy blind signature has the following proprieties. [38].

- Verifiability: The proxy blind signature can be correctly verified by the verifier.
- Unforgeability: No entity except for the proxy signer can generate a valid proxy blind signature.
- Identifiability: Any entity can know the identities of both the original signer and the proxy signer.
- Nonrepudiation: Both the original signer and the proxy signer cannot later deny the generation of the signature.
- Unlinkability: After the requester has deduced the unblinded version signature for verification, the proxy signer (or the original signer) is unable to link the blind signature and the deduced signature.

3.5.3.1 Delegation by Warrant Algorithm

The original signer (O) signs the warrant with their Elliptic Curve Proxy Private Key ($EPPR_O$). To sign the warrant, the original signer performs the following steps.

a. The original signer randomly chooses a number k_1, where $1 < k_1 < n$ and computes the following: $R_1 = K_1 * \mathbb{P}$. This gives the following points (x_1,y_1). Then the original signer does the following: $r_1 = x_1 \mod n$.

b. The original signer generates a signature on the warrant (Sig_{m_w}) using the key ($EPPR_O$). As seen in Equation (1.1)

$$Sig_{m_w} = EPPR_O + k_1 * h(m_w||r_1) \mod n. \tag{3.1}$$

c. The original signer delivers the delegation message (R_1, Sig_{m_w}, m_w) to the proxy signer through a secure channel.

d. Once received by the proxy signer, the proxy signer first checks the following equation. If the equation holds the proxy signer moves to step (e).

$$Sig_{m_w} * \mathbb{P} = R_1 * h(m_w||r1) + y_0. \tag{3.2}$$

e. The proxy signer generates both the proxy public and private key ($EPPK_p$,$EPPR_p$) respectively by using the following equations.

$$EPPR_p = EPR_p + Sig_{m_w} \mod n. \tag{3.3}$$

$$EPPK_p = EPK_o + EPK_p + R_1 * h(m_w||r_1). \tag{3.4}$$

The proxy signer randomly chooses a k_p, where $1 < k_p < n$ and computes $R_p = K_p * \mathbb{P}$. This will give the following point: (x_2,y_2). Then the proxy signer does the following. $r_p = x_2$ mod n. It then sends the following values to the sender who aims to use proxy blind signature (R_1, R_p, m_w).

3.5.3.2 Blind Message (BlndMsgGen) Algorithm

Here a sender wants to use the proxy blind signature service provided by the proxy signer. The sender first generates a blinding value (R*) using Equation (3.5), where a, b, and c are random blinding factors. It then computes a hash value of the message (m) using Equation (3.6). It then blinds the hash value e* using Equation (3.7). After that it sends a request for a blind signature on the hash value to the proxy signer.

$$R^* = a * R_p + c * \mathbb{P} - b * EPPK_p \tag{3.5}$$

$$e^* = h(R^*||m) \tag{3.6}$$

$$e = a^{-1}(e^* - b) \mod n \tag{3.7}$$

3.5.3.3 Blind Signature Generation (BlndSigGen) Algorithm

Once the request is received by the proxy signer, it generates the blind signature (S^*) on (e) using Equation (3.8). Then it sends the blind signature (S^*) to the sender.

$$S^* = e * EPPR_p + k_p \mod n \tag{3.8}$$

3.5.3.4 Blind Signature Driven (BlndSigDrv) Algorithm

After receiving the blind signature (S^*) from the proxy signer, the sender derives the unblind version of signature (S) using Equation (3.9). This signature (S) can be proven by any verifier using Equation (3.10).

$$S = S^* a + c \mod n \tag{3.9}$$

3.5.3.5 The Blind Signature Verification (BlndSigVer) Algorithm

This algorithm is executed by a verifier. After receiving the blind signature, the verifier verifies the proxy blind signature using Equation (3.10).

$$e^* = h((S * \mathbb{P} - e^* EPPK_p)||m) \tag{3.10}$$

3.6 Digital Certificate

A digital certificate [39] is a data structure that is generated and signed by a trusted authority called a Certificate Authority (CA). The data structure fields specified in a standard certificate are defined by the X.509 Certificate Specification [39]. These fields are as follows:

- Version: This field describes the version of the certificate.
- Serial Number: A unique number issued by the CA to each certificate.
- Signature: An identifier for the algorithm used by the CA to sign the certificate.
- Issuer: The entity that has signed and issued the certificate.
- Validity: The certificate validity period is the time interval during which the CA permits that it will maintain information about the status of the certificate. This field contains two dates: the date on which the certificate validity period begins and the date on which the certificate validity period ends.
- Subject: The owner of the public key and the certificate.
- Subject Public Key Info: This field is used to carry the public key and identify the algorithm with which the key is used (e.g., RSA or ECC).

- Unique Identifiers: This field presents the unique identifiers of the subject and issuer to handle the possibility of reuse of subject and/or issuer names over time.
- Extensions: This field provides methods for associating additional attributes with users or public keys and for managing relationships between CAs.
- Signature: This field contains the issuer's signature on the data fields and details about the signature algorithm.

3.7 Chapter Summary

This chapter has presented the cryptographic building blocks which are used in the design of our novel methods and protocols which will be presented in the upcoming chapters.

Chapter 4

SPDC Framework: An Architecture Design

4.1 Chapter Introduction

This chapter describes our innovative framework, called the Secure and ID-Privacy Preserving Distributed Data Collection (SPDC) framework. It describes the SPDC architecture, entities, and the interaction among these entities. The chapter first presents an initial design of the framework architecture called the First Data Collection (FDC) system. The FDC system fails to satisfy several design requirements, in addition to raising some challenges which motivated us to design the SPDC framework architecture.

In detail, the chapter is organized as follows. Section 4.2 describes the design of our first data collection (FDC) system. Section 4.3 analyzes the FDC system based on the design requirements identified in Chapter 2. Section 4.4 describes the high-level ideas used to design the SPDC framework. Section 4.5 describes the SPDC architecture, its entities, the components of each entity and the interactions among these components. Section 4.6 analyzes the SPDC framework against our designed requirements. Section 4.7 discusses the SPDC framework and its features. Finally, in Section 4.8, a summary of the chapter is provided.

4.2 First Data Collection (FDC) System

In Chapter 2, we proposed some recommendations to design a secure data collection system. Among these recommendations was a new architecture that consists of different service providers and a healthcare provider. We pointed out that using different service providers to collect a patient's data could prevent the inference attack. In this section, we provide the design for the first version of the architecture and we call it the first architecture of the data collection (FDC) system. The FDC system architecture consists of the Healthcare Provider (HCP), different data collection servers, and patients.

4.2.1 Challenges

Accessing distributed data collection servers owned by different service providers requires authentication using one's true identity that is to ensure that the patient is authorized to use the collection service. However, the true identity of the patient should be protected from being revealed to any entity except the HCP. One solution to access distributed servers without revealing one's true identity is to design the FDC system as a federated identity management authentication systems (e.g. institutional sign-in at IEEE Xplore)[40]. The federation system consists of identity providers, service providers and users. The identity provider is an entity that creates, manages, and stores users' digital identities for authentication. The service provider is an entity that provides the user with the requested service.

The federation system works as follows. Each time the user wishes to use a service provided by a service provider, the user is redirected to the identity provider by the service provider to perform authentication. The identity provider receives the authentication request and returns an accept or reject assertion to the service provider based on the user status in the system. The assertion is a statement that the user has either been granted or denied access. In the FDC system, we consider the HCP as the identity provider of the system, while data collection servers as service providers.

4.2.2 Assumptions

Here we list our assumptions before explaining the FDC system and its interaction.

- The only trustworthy and honest entities are the HCP and the patient.
- The patient has registered with the HCP using his/her real ID. After the registration, the patient obtains a credential (e.g. username and password) from the HCP.
- A data collection server has registered with the HCP to participate in collection service. After the registration, the data collection server has obtained a digital certificate.
- A mobile agent, run on the patient's mobile device, randomly selects a set of servers each day to upload the data. The patient can also select the servers manually.
- The data uploaded on any server is encrypted and signed using a secret key known only to the patient and the HCP.

4.2.3 FDC System Architecture

The FDC system architecture (as shown in Figure 4.1) consists of the following:

- Patients: Each patient wears small sensors to measure and monitor the patient's health data (e.g. ECG, blood pressure). The health data is sent to the patient's mobile device (i.e., via

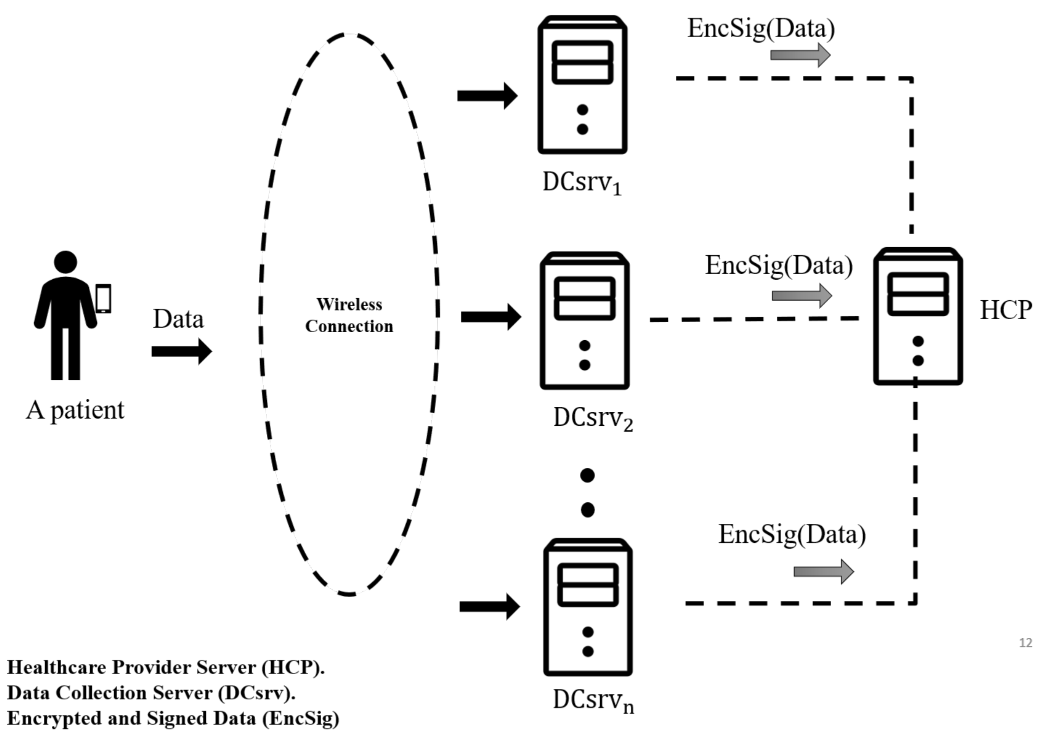


Figure 4.1: FDC system architecture.

Bluetooth) for processing. The mobile device classifies the data into normal or urgent data. Then, the mobile device encrypts, signs and sends the encrypted and signed data (EncSig-Data) to a data collection server. For the sake of simplicity, in the remaining part of the thesis, we will use the terms ’patients’ and ’mobile devices’ interchangeably to refer to the same entity.

- Data Collection Servers (DCsrvs): These servers are owned by different service providers (e.g. Google, Amazon, Yahoo). The main function of these servers is to collect and store the patient’s data temporarily and then forward the data to the healthcare provider (HCP). We assume an unlimited number of service providers and that they are geographically distributed around the world.

- HealthCare Provider (HCP): This entity runs the business logic of the system. It is the ultimate destination for the patient’s data. The HCP receives the encrypted and signed data from different data collection servers. The HCP verifies, analyzes and stores the data. In addition, the HCP is the identity provider. This means that any entity (i.e. a patient or a data collection server) wanting to participate in the system should first register with the HCP.

4.2.4 FDC System Interaction

The system interaction of the FDC system has two phases: authentication and uploading. In the authentication phase, the patient should be authenticated to a data collection server through the HCP before uploading. In the subsequent uploading phase, the patient should start uploading.

4.2.4.1 Authentication Phase

As we stated in the assumption section, the mobile agent randomly selects a number of servers to use to upload the data. For simplicity, we assume that the patient has selected only two data collection servers, DCsrv1, and DCsrv2. In this phase, the patient first obtains an authentication code (AuthCode) from the HCP, then sends the AuthCode to the server (DCsrv1 or DCsrv2) to gain access to the collection service. The AuthCode contains the following information about the patient, the pseudonym ID generated by the HCP to be used with the data collection server, the expiration date of the AuthCode, and the signature of HCP on the AuthCode. The authentication phase, as shown in Figure 4.2, involves the following steps.

(1) The patient sends a service request (uploading service) to a data collection server (DCsrv1).

(2) The DCsrv1 responds to the patient by redirecting the patient to the HCP server. The redirection message includes a request for an authentication code (AuthCode) that can be used to identify the patient.

(3) The patient sends an authentication request to the HCP.

(4) The HCP challenges the patient to provide a valid credential.

(5) The patient provides the credential, identifying him/her self with a patient ID and password.

(6) The HCP validates the patient credential. If the HCP accepts the credential, the HCP grants the patient the AuthCode.

(7) The HCP sends the AuthCode to the patient.

(8) The patient forwards the AuthCode to the DCsrv1.

(9) The DCsrv1 receives the AuthCode from the patient and then verifies the HCP signature on the AuthCode. After that, it creates a local account for the patient with the pseudonym ID stated in the AuthCode.

(10) The DCsrv1 grants access to the patient.

(11) The patient starts uploading. Later we will see that the uploaded data will be forwarded to the HCP by each data collection server.

(12) When the AuthCode expires, the patient needs to repeat steps 1 to 10.

We have here shown the authentication on the first server (DCsrv1). The authentication on the second server (DCsrv2) follows the same steps.

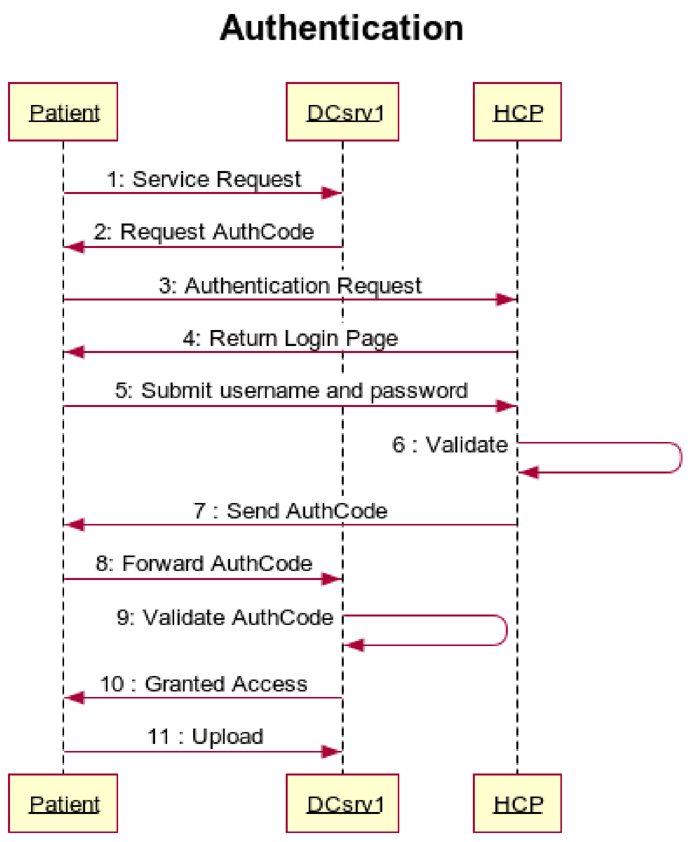


Figure 4.2: The authentication phase

4.2.4.2 Uploading Phase

After the patient has been authenticated with both servers (i.e., DCsrv1 and DCsrv2) as explained in the authentication section, the patient starts uploading to the DCsrv1. The uploading phase, as shown in Figure 4.3, can be explained as follows.

(1) The patient's mobile device collects the health data from wearable sensors, classifying it as normal or critical. Then, it uses the secret key (obtained from the HCP) to generate the MAC on the patient-generated health data (PGHD) then encrypts both the MAC and the PGHD together. This type of data is called an encrypted MAC and patient generated health data (EMPGHD). It then constructs a message (Msg) that contains the EMPGHD, the priority of the data (urgent/normal), and the authentication code (AuthCode) which is used to identify the patient to the server.

(2) The patient then sends the message to the DCsrv1.

(3) Upon receipt, the DCsrv1 first verifies the AuthCode of the patient and checks that it has not expired. It then checks the priority of the data. If the priority is normal, it stores the encrypted data under the patient's pseudonym. Otherwise, it sends an urgent message to the HCP.

(4) The DCsrv1 sends an acknowledgment message to the patient to confirm that the data has been uploaded successfully.

(5) After finishing uploading to the DCsrv1, the mobile device has the option to continue uploading on the same server (DCsrv1) or breaking and uploading to the second one (DCsrv2). This decision must be accomplished in a way that no entity inside or outside the system can learn the pattern for the selection of the server and the pattern of uploads.

(6) The patient decides to send the second uploading message to the DCsrv2.

(7) Upon receipt, the DCsrv2 first verifies the AuthCode of the patient and checks that it has not expired. It then checks the priority of the data. If the priority is normal, it stores the encrypted data under the patient's pseudonym. Otherwise, it sends an urgent message to the HCP.

(8) The DCsrv2 sends an acknowledgment message to the patient to confirm that the data has been uploaded successfully.

(9) Both servers, DCsrv1 and DCsrv2, send the patient data to the HCP. Each data collection server aggregates the patient data as follows. It adds each encrypted datum of the patient to the patient's pseudonym in one data structure. Then, it combines multiple patients' data in one packet.

(10) The DCsrv1 sends the patients' aggregated health data to the HCP.

(11) Upon receipt, the HCP uses each patient's pseudonym to look up the corresponding secret key in the database. It then uses the patient's secret key to decrypt the data and uses the same key to verify the authenticity of data by verifying the MAC. The HCP executes the same process (i.e., looking up a pseudonym and secret key to conduct authentication) for all registered patients.

4.3 FDC Analysis

4.3.1 Requirements Analysis

In this section, we analyze the FDC system against our designed requirements stated in Chapter 2.

- FDC supports two modes of data collection: The FDC system supports periodic and on-event data collection modes. As one can see in the uploading phase, the patient can upload periodically to any data collection server. When the data collection server receives the uploading message labeled with high priority, the data collection server sends an urgent message directly to the HCP.

- FDC preserves a patient's ID privacy: The FDC system supports ID anonymity as none of the data collection servers can learn the real identity of a patient. Each data collection server identifies the patient under different pseudonyms generated by the HCP. However, the FDC

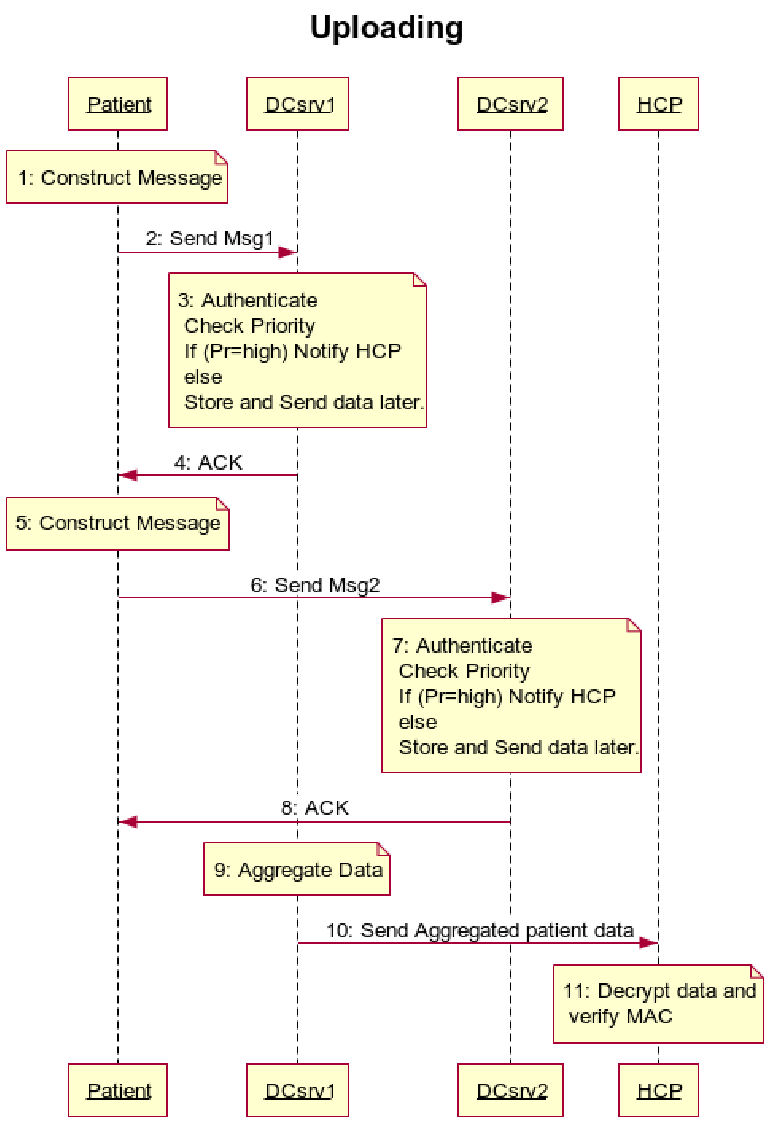


Figure 4.3: The uploading phase

system fails to protect session unlinkability as the patient may upload multiple times to the same server using the same pseudonym for each upload session.

- FDC supports entity authentication: As we explained in the authentication phase, when the patient requests an uploading service from a data collection server, the server first redirects the patient to conduct the authentication via the HCP. The HCP challenges the patient to provide the correct credential. Upon successful verification, an AuthCode is returned to the patient and the patient sends the AuthCode to the data collection server. The FDC system supports the design requirement S1 (i.e. mutual entity authentication). It fails to protect against impersonation attacks. This is because the AuthCode is susceptible to being stolen by an adversary. The adversary can reuse the AuthCode to request access to the data collection server.

- FDC provides an end-to-end data authenticity: In the FDC system, the patient receives a secret key from HCP to generate MAC on plain data. Then, the patient encrypts both the plain data and the MAC using the same secret key. When the patient uploads the encrypted data on a data collection server, the server will not be able to learn anything about the patient's data.

Only the HCP which knows the secret key can decrypt the data and check its authenticity by verifying the MAC.

- FDC provides end-to-end data confidentiality: In the FDC system, the confidentiality of data is protected through the encryption mechanism.
- FDC supports scalability: The data collection servers are used to collect data from a number of patients anywhere and anytime. However, it fails to reduce the number of communication messages. This is because, in the authentication phase, the patient needs to be redirected to the HCP to perform the authentication first.

4.3.2 Discussion

The FDC system does not satisfy all the design requirements and has both a number of security and performance issues, these issues can be explained as follows.

A. HCP is a single point of failure: The HCP is responsible for linking each patient's data (which is identified under different pseudonyms) to the patient's real identity. This function involves the awareness of the HCP of each data collection server a patient uses and each pseudonym the patient is using with each data collection server. Relying on a single entity to perform the linkability function for all patient's data could create a single point of failure and also threaten the privacy of the patient.

B. The system is not patient-centric: The patient is not responsible for generating the pseudonyms and credentials to access data collection servers. The patient is reliant on the HCP to generate pseudonyms for each data collection server and conduct the authentication on behalf of the patient.

C. Authentication delay: The authentication suffers from a round-trip delay. This is because the HCP is used to perform the authentication on behalf of the patient. Each data collection server forwards an upload request coming from each patient, to the HCP, to verify the authenticity of the patient.

D. Authentication threats: The adversary may try to impersonate a patient to gain unauthorized access to the patient's data. The adversary can do this by simply stealing the AuthCode. Such impersonation attacks are sometimes also referred to as identity theft in the literature.

In the next section we introduce the architecture of our innovative framework. This framework is called the Secure and ID-Privacy Preserving Distributed Data Collection (SPDC). It is designed to overcome the challenges identified in the FDC system.

4.4 SPDC a Novel Framework

To further improve the FDC system and satisfy the design requirements mentioned in Chapter 2, we designed the SPDC framework. The design of the framework should satisfy the design requirements mentioned in Chapter 2 and the ones listed below.

4.4.1 Additional Requirements

The SPDC framework should support the following additional requirements.

- Patient-Centric (F2): The patient should be involved in the generation of the pseudonyms and credentials.
- Prevent impersonation attack (S4): The framework should be protected against impersonation attack.
- Enhance efficiency: The efficiency of the framework should be enhanced further by (E2) applying the principle of load distribution and separation of duties among entities, and (E3) the authenticating should be done without imposing a further delay (i.e., without redirection).

4.4.2 High Level Ideas

Idea 1. Design hierarchical servers to satisfy the ID privacy and efficiency requirements and to solve issue A: Instead of using all the data collection servers to upload the health data, the data collection servers should be divided into a home server and foreign servers. The details are as follows:

- The patient has the right to select a set of servers, but this time the patient should assign one of them as the home server, and the others are considered as foreign servers. The patient will use foreign servers for data uploading.
- The home server allows the patient to access foreign servers and be the patient's anonymous linkage server. This means that the home server is responsible for collecting the patient's encrypted data which is scattered across multiple foreign servers and then sending the collected data to the HCP.
- This division is not fixed; it is dynamic (e.g. changed every 24 hours). This means that one of the patient's foreign servers can be the home server the day after. The foreign servers which were used by the patient to collect the data will not be used again until a period has passed (e.g. after 30 days). Then they will be reused again to collect the patient's data. Changing servers each day may increase the privacy protection level.
- One can think of these servers as having three layers. The first is multiple foreign servers collecting health data from the patient. The second layer is the home data collection

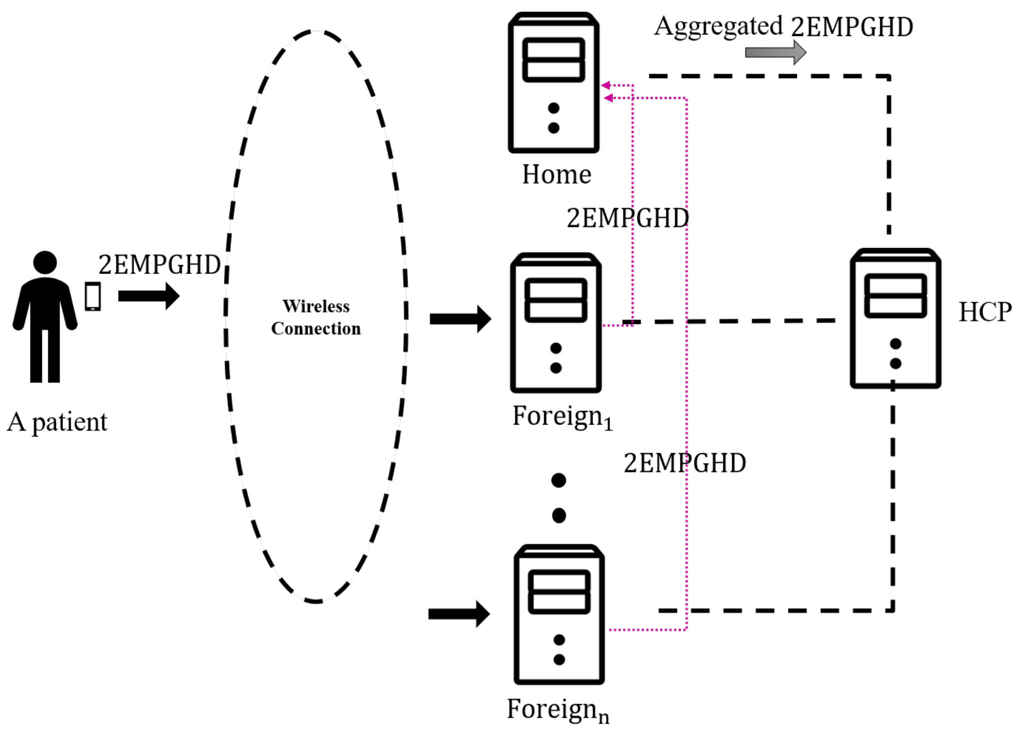


Figure 4.4: The SPDC system architecture

server linking and aggregating the patient data anonymously from different servers. The third layer is the HCP which is the ultimate final storage for all the patient's health data. To implement this idea a new system architecture is designed as shown in Figure 4.4.

Idea 2. Design multiple index pseudonyms and multiple request pseudonyms to satisfy ID privacy preservation requirements, solve issue B, and satisfy the requirement E2: For each patient, data is collected by foreign servers, and the collection process with one server may be executed several times (periodically). Therefore, preserving the patient's ID privacy requires the following:

- First, with each server, the patient should have a pseudonym ID called the 'index pseudonym'. The index pseudonym serves as an account name. This is to prevent multiple servers from colluding together to identify the patient.
- Second, for each request with the same server, the patient should use a fresh pseudonym called request pseudonym; this to solve issue E2.
- There must be a relation between the index and the request pseudonyms. The relation can be explained as follows. The request pseudonym should be generated based on the index pseudonym. This is to allow the server to link each upload request generated by the same patient to the patient's account.
- The details designed of the methods used for pseudonyms generation and linkage are discussed in Chapter 5.

Idea 3. Design a method to satisfy the security requirements (S2, S3): This method enables the home and any foreign server to verify the authenticity of the patient's data without learning the content of the data. This method can be explained as follows.

- Before the patient uploads his/her data on any foreign server, s/he should first generate a MAC on the health data using a secret key and then encrypt the data and the MAC using the same key. This process should be performed twice.
- The first time, the patient generates a MAC on plain data and then encrypts both the plain data and the MAC using the secret key which is only known to the patient and the HCP. The result of this first process is called an encrypted MAC and patient-generated health data (EMPGHD).
- The second time, the patient generates the MAC in the EMPGHD and encrypts both the second MAC and the EMPGHD using another secret key. This secret key is known only to the patient's home server and the patient. This method is used to prevent any foreign server from learning the content of the patient's data yet allows the home server to verify the authenticity of the data by verifying the second MAC. The details designed of this method are discussed in Chapter 7.

Idea 4. Design an anonymous authentication method to satisfy the security requirements and to solve issue C: To ensure authorized use of the data collection service, the patient should be identified and authenticated before they are allowed to access the collection service and this should be achieved without compromising the patient's ID privacy. In addition, we need to reduce the round-trip delay imposed by forwarding the authentication request to the HCP. To support such authentication in a seamless and scalable manner, we have designed two anonymous authentication credentials as follows. The detailed explanation is provided in Chapter 6.

- A home pseudonym certificate (HCert) for authenticating the patient with the home server and foreign pseudonym certificates (FCerts) for authenticating the patient with foreign servers.
- The HCert is generated by the HCP to allow the patient to register with any server as a home server.
- The FCert is generated in cooperation between the patient and the selected home server using the blind signature mechanism.
- The mobile device of the patient generates the fields of FCert and blinds the FCert so the home server does not know anything about the content of FCert. The mobile device then sends the certificate to the home server. The home server blindly signs the certificate and sends it back to the patient who unblinds the signature, then attaches the signature on the FCert. The certificate is now ready to be used.

Idea 5. Apply data aggregation and the principles of distributed load-sharing and separation of duties to satisfy the efficiency and scalability requirements of E: Patients' data are classified into two categories: normal data and abnormal data. Normal data are aggregated by the home server of each patient before forwarding them to the final destination (i.e., HCP). This is to save bandwidth costs. Abnormal data are forwarded by the home server of the patient to the HCP without further processing. This is to reduce any latency so that data can reach the HCP as soon as possible. In addition, the linkage of the health data of each patient which are scattered across multiple foreign servers is carried out by the patient's home server. Moreover, the generation and linkage of pseudonyms and the corresponding certificates are carried out by multiple entities. The detailed explanation is provided in Chapter 5, 6 and 7.

4.5 SPDC Framework Architecture

To implement the above ideas we need to re-design the FDC architecture presented in Figure 4.1. The SPDC architecture as seen in Figure 4.4 consists of the same entities in the FDC system. However, the data collection servers are divided into home and foreign servers. This division is from a patient's perspective. This means that a home server for one patient can be a foreign to another patient. Note that we are using the same assumptions mentioned in 4.2.2.

4.5.1 Architecture in Detail

Here are the entities of the SPDC framework.

- The Patient: The patient wears small devices integrated with low-power computation, communication and storage modules to measure and monitor health data (e.g. ECG, blood pressure) from the patient. The health data will be sent to a mobile device (i.e. via Bluetooth) for processing. The mobile device classifies the data into normal or abnormal data, and structures the data into a predefined format, called Patient-Generated Health Data (PGHD). The PGHD is uploaded to foreign servers.

- Data collection servers DCsrvs: Data Collection servers (DCsrvs) are servers that are used to collect, store, aggregate, and deliver the collected data to the healthcare provider (HCP). These servers are owned by different service providers. Each data collection server plays two roles: home and foreign servers. This means that the data collection server may act as a home server for one patient, but a foreign server to another. Each patient registers with one server which is called the patient's home server (it can be also called the anonymous linkage point). The other servers are called the patient's foreign servers. The home server allows the patient to access foreign servers by blindly signed certificates generated by the patient, one for each server. The foreign servers are used by the patient to upload the health data. Then, each foreign server forwards the health data to the patient's home server. It should be noted that the home and foreign servers for each patient are not fixed but changeable each day. The selection of the foreign and home servers is out of the scope of this thesis.

  – In summary, the tasks performed by a home server are: (i) anonymously linking data for the registered patients which arrive from different foreign servers, (ii) issuing a blind signature on certificates generated by the patient to access foreign servers, and (iii) aggregating the normal data received from the registered patients before delivering the data to the HCP and immediately forwarding the urgent data to the HCP.

  – The tasks performed by the foreign servers are: (i) collecting data from patients, and (ii) forwarding the urgent data without aggregation to the respective home servers of the patients, while aggregating normal data before forwarding it.

- The Healthcare Provider Server (HCP): This runs the business logic of the system. It is the ultimate destination for all patients' data. It is where this data is stored, processed, and analyzed. The HCP receives the patients' data from their respective home servers.

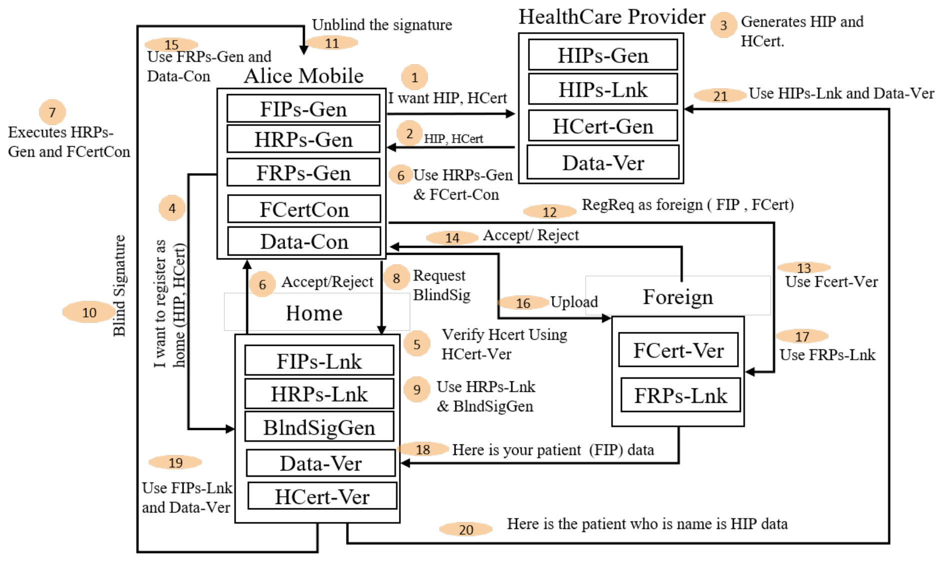
4.5.2 Entity Components

Each entity in the framework has a number of components. The following presents the components for each entity and their function. These components are fully described in chapters 5, 6 and 7.

- The HCP consists of the following components: home pseudonym certificate generation (HCert-Gen), home index pseudonym generation (HIP-Gen) and linkage (HIP-Lnk), and data verification (Data-Ver). The HIP-Gen component is used to generate a home index pseudonym based on the patient real ID, while the HIP-Lnk is used to link the home index pseudonym back to the patient real ID. The HCert-Gen component generates a home pseudonym certificate that verifies the patient's home index pseudonym. The Data-Ver component verifies that the patient's data is generated by the claimed patient and has not been modified between uploading and reaching the HCP.

- The mobile device belonging to the patient contains of the following components: foreign index pseudonyms generation (FIPs-Gen), home request pseudonym generation (HRPs-Gen), foreign request pseudonym generation (FRPs-Gen), foreign certificate construction (FCert-Con), and data construction (Data-Con). The function of the FIPs-Gen component is to generate a fresh foreign index pseudonym for each foreign server. The function of the HRPs-Gen component is to generate a fresh home request pseudonym for each request sent to the home server. The function of the FRPs-Gen component is to generate a fresh foreign request pseudonym for each uploading request sent to a foreign server. The function of the FCertCon component is to generate a fresh certificate for each foreign server. This component consists of a number of mini-components as follows: blind message generation (BlndMsgGen), blind signature verification (BlndSigVer), and unblinding signatures (UnBlndSig). The function of the Data-Con component is to generate MAC and encrypt patient data using two keys (as explained in idea 3).

- The data collection server consists of a number of components. Since each data collection server plays a role of both home, for some patients, and foreign for other patients, there



HealthCare Provider
HIPs-Gen
HIPs-Lnk
HCert-Gen
Data-Ver
3 Generates HIP and HCert.
21 Use HIPs-Lnk and Data-Ver
Alice Mobile
FIPs-Gen
HRPs-Gen
FRPs-Gen
FCertCon
Data-Con
1 I want HIP, HCert
2 HIP, HCert
6 Use HRPs-Gen & FCert-Con
11 Unblind the signature
15 Use FRPs-Gen and Data-Con
7 Executes HRPs-Gen and FCertCon
4 I want to register as home (HIP, HCert)
10 Blind Signature
12 RegReq as foreign (FIP , FCert)
14 Accept/ Reject
13 Use Fcert-Ver
6 Accept/Reject
8 Request BlindSig
16 Upload
Home
Foreign
5 Verify Hcert Using HCert-Ver
9 Use HRPs-Lnk & BlndSigGen
17 Use FRPs-Lnk
FIPs-Lnk
HRPs-Lnk
BlndSigGen
Data-Ver
HCert-Ver
FCert-Ver
FRPs-Lnk
18 Here is your patient (FIP) data
19 Use FIPs-Lnk and Data-Ver
20 Here is the patient who is name is HIP data

Figure 4.5: SPDC Framework Interactions

are some components used for home patients and others for foreign patients. The components for home patients are: home request pseudonyms linkage (HRPs-Lnk), foreign index pseudonyms linkage (FIPs-Lnk), home pseudonym certificate verification (HCert-Ver), blind signature generation (BlndSigGen), and data verification (Data-Ver). The HRPs-Lnk component links each home request pseudonym generated by the patient to the home index pseudonym to identify the patient. The FIPs-Lnk component links each foreign index pseudonym generated by the patient to the patient's home index pseudonym. The BlndSigGen component signs each foreign pseudonym certificate generated by the patient blindly and returns it to the patient. The Data-Ver component verifies the authenticity of the patient's data using the home shared key (HSK). The components for foreign patients are: foreign request pseudonym linkage (FRPs-Lnk) and foreign pseudonym certificate verification (FCert-Ver). The FRPs-Lnk component is used to link each foreign request pseudonym generated by the patient to the patient's foreign index pseudonym. The FCert-Ver component is used to verify the foreign pseudonym certificate of the patient.

4.5.3 Interaction in a Nutshell

Here we show how each entity in the system interacts with each other. There are eight phases: (1) the registration with the HCP, (2) selecting the home and foreign servers, (3) registration with the home server, (4) generating the foreign pseudonym certificate in cooperation between the home and patient, (5) registration with the foreign servers using the generated foreign pseudonym certificates one for each server, (6) uploading the patient's data to multiple foreign servers, (7) forwarding the patient's data from the foreign servers to the patients' respective home servers, and (8) delivering the patients' data from their home servers to the HCP. These phases are illustrated in Figure 4.5.

4.5.3.1 Phase 1: Registration With HCP

- In step (1), the patient sends a registration request to the HCP asking the HCP to generate a home index pseudonym and home pseudonym certificate.
- In step (2), upon receipt of the registration request, the HCP generates a home index pseudonym using the HIP-Gen component and a home pseudonym certificate using the HCert-Gen component. Then it sends them to the patient.

4.5.3.2 Phase 2: Selecting Home and Foreign Servers

Each day the mobile device should select a number of servers, one of which should be a home server and the others foreign servers. The selection should be done as follows. The foreign servers cannot be re-used as upload servers until a time has passed, such as 30 days. This means that each day the patient should have a new set of foreign servers. Any foreign server can be the home server. The home server can be reused after a period has passed (e.g. after 30 days). The detailed mechanism for how the selection is made is out of the scope of the thesis. We here only assume that there is an unlimited number of data collection servers to collect and anonymously link the patient's data until it reaches its final destination.

4.5.3.3 Phase 3: Registration With Home server

- In step (4), the patient sends a registration request to the home server. The request message carries the following information: the home index pseudonym of the patient and the home pseudonym certificate.
- In step (5), upon receipt of the registration request, the home server validates the HCP signature on the home pseudonym certificate using the HCert-Ver component. If the home accepts the registration request, it sends the response message to the patient. The response message can be either 'accept' or 'reject'.

4.5.3.4 Phase 4: FCert Construction

The FCert is generated by the patient and blindly signed by the home. This certificate is used to register with a foreign server, with one certificate for each server. We suggested using a blind signature because we do not want the home servers to learn anything about the certificate and where the patient is going to use this certificate. The details of this phase are provided in Chapter 6.

- In step (7), the patient generates an FCert certificate which consists of information about the patient such as a foreign index pseudonym which is generated using the FIP-Gen. Then the patient generates a home request pseudonym using HRPs-Gen. Subsequently, the patient blinds the certificate using the BlndMsgGen component and sends a blind signature request message to the home server. The message carries the blind FCert, and the HRP. The HRP is used to identify the patient for the request.

- In step (9), the home server first links the home request pseudonym using the HRP-Lnk back to home index pseudonym to identify the patient. Then, it generates the blind signature using BlndSigGen with its private key and returns the signature to the patient.

- In step (11), upon receipt, the patient derives the unblind version of the signature. Then, the patient attaches the signature to the FCert certificate and stores the certificate to be used later.

4.5.3.5 Phase 5: Registration With a Foreign Server

- In step (12), the patient sends a registration request to the foreign server carrying the foreign index pseudonym of the patient and the foreign pseudonym certificate. Then it sends the request to the foreign server.

- In step (13), upon receipt of the registration request, the foreign server validates the home signature on the foreign pseudonym certificate using the FCert-Ver component. Upon verification, the foreign server stores the information in database entries.

- In step (14), the foreign server sends the response message to the patient.

4.5.3.6 Phase 6: Uploading With a Foreign Server

This phase is executed between the patient and a foreign server. It allows the patient to upload data to the foreign server using a swarming algorithm. The details of this phase are provided in Chapter 7.

- In step (15), the patient first generates a MAC and encrypted data using the Data-Con component. The mobile device generates the first message authenticated code (MAC) on the plain patient-generated health data (PGHD) using the shared key (SK) which is known only to the patient and the HCP. The result of this process is called the EMPGHD. After that, the patient generates the second MAC on the EMPGHD using the home shared key (HSK), this key is the patient shared key with the home server. Then it encrypts both MAC2 and the EMPGHD using the same key (i.e., HSK). This results in 2EMPGHD which will be uploaded to the foreign server.

- In step (16), the patient generates the fresh foreign request pseudonym using FRPs-Gen. After that, the patient sends an upload request to the foreign server. This request message carries the fresh foreign request pseudonym and the patient's data.

- In step (17), the foreign server receives the upload request, which first links the FRP to the FIP using the FRPs-Lnk. Then, it checks the priority of the patient data: if it is high priority then the foreign server sends a notification directly to the patient's home server, then the home server will notify the HCP. Otherwise the foreign server stores the data and later on forwards them to the patient's home server.

4.5.3.7 Phase 7: Forwarding

This phase is executed between data collection servers. Each data collection server forwards patients' data to their respective home servers. The details of this phase are provided in Chapter 7.

- In step (18), the foreign server aggregates each patient's data and sends the aggregated data to the patient's home server.
- In step (19), the home server receives the data. It verifies the authenticity of the data using Data-Ver and the home shared key (HSK).

4.5.3.8 Phase 8: Delivery

This phase is executed between a data collection server and the HCP. It allows the data collection server to send the patient's data to the final destination. The details of this phase are provided in Chapter 7.

- In step (20), the home server starts by aggregating each patient's data under the home index pseudonym and then constructing an aggregated message that holds a number of patients' data. Then, the home server sends the message to the HCP.
- In step (21), the HCP receives the message, and first extracts each patient's data. It then uses the HIP-Lnk to link each HIP to a patient's real ID and then finds the shared key for the patient to verify the authenticity of the data using Data-Ver. After that it stores the patient's data in the database.

4.5.3.9 Changing Home and Foreign Servers

The patient can change the home and foreign servers. To change the home server the patient requests a new home index pseudonym (HIP) and a corresponding home pseudonym certificate (HCert) from the HCP. The patient uses the HCert to register with any server as a home server. After the registration with the home server, the patient can request a number of FCerts to register with a number of foreign servers.

4.6 Design Requirement Analysis

In this section we analyze the SPDC framework against our design requirements.

- The SPDC supports functional requirements: SPDC supports various modes of data collection (F1) namely periodic and on-event data collection modes. As can be seen in the uploading phase, the patient can upload periodically to any foreign server. When the foreign server receives an upload message labeled with high priority, it sends an urgent message directly to the home server which immediately sends a notification message to the HCP. The SPDC is also patient-centric (F2) as the patient is responsible for generating the pseudonyms and credentials.

- The SPDC preserves a patient's ID privacy: The SPDC system supports the ID anonymity propriety as none of the data collection servers can learn the real identity of the patient except the HCP. Each data collection server identifies the patient under different pseudonyms generated by the HCP or by the patient. The home server identifies the patient under a home index pseudonym. The same patient is identified under different foreign index pseudonyms with different foreign servers. Each session with any server is identified under a fresh pseudonym. In addition, the SPDC is not only protecting the patient's ID from being revealed but it also makes sure that other attributes that can be used to identify the patient are also protected. One of these attributes is the upload pattern. The SPDC protects the upload pattern from being revealed by using different data collection servers to collect the patient's data and encouraging the patient to select a new set of servers each day. Thus we can say that the SPDC protects the patient IDs from being linked across domains of multiple servers, across multiple sessions with the same server, and protects the upload pattern from being inferred by other servers. The SPDC preserves patient's real ID anonymity (P1), supports unlinkability (P2), and prevents attribute disclosure (P3).

- SPDC supports anonymous authentication: The SPDC system supports anonymous authentication using pseudonym certificates. As we explained in the authentication phase, the patient is authenticated with the home server first using a home pseudonym certificate issued by the HCP. Then the patient requests a number of blind signatures on a number of foreign pseudonym certificates generated by the patient. These certificates will be used to authenticate the patient with each foreign server. In addition, there is no delay in the authentication as a result of redirecting authentication to the HCP as seen in the FDC system. Moreover, there is no impersonation attack as the adversary needs to prove knowledge of the private key corresponding to the public key stated in the certificate. Therefore, we can say that the SPDC supports mutual authentication (S1), prevents impersonation attacks (S4), and enhances the authentication by preventing redirection (E3).

- SPDC provides end-to-end data authenticity (S2): In the SPDC, the patient's data is protected using the sign-then-encrypt method. In this method, the patient has two secret keys: one key is known to the patient and the HCP, and the other is known to the patient and the home server. As we explained in the uploading phase, before the patient uploads his/her data to any foreign server, s/he should first generate a MAC on plain data and then encrypt both the plain data and the MAC using the first secret key which is only known to the HCP and the patient. The result of this first process is called encrypted MAC and patient-generated health data (EMPGHD).

Second, the patient generates a second MAC on the EMPGHD and encrypts both the second MAC and EMPGHD using the second key known as the home shared key. The result of this second process is called 2EMPGHD. When multiple 2EMPGHD for the same patient arrives from multiple foreign servers to the patient's home server, the home server uses the patient's home shared key to verify the authenticity of the data before sending it to the HCP. Finally, the HCP verifies the authenticity of the plain data using the first secret key which is known to the HCP and the patient.

- SPDC provides end-to-end data confidentiality (S3): In the SPDC, the confidentiality of any patient's data is protected from any other data collection servers using encryption. Only the HCP can learn the content of the data.

- SPDC provides efficiency: The SPDC uses the mechanism of aggregation, the principles of load distribution, and the separation of duties. As one can see, the pseudonym generation and linkage beside the credentials generation are distributed among entities. The patient generates request pseudonyms to be used for each request with a data collection server, while the data collection server links the request pseudonym to the corresponding index pseudonym. The index pseudonyms are generated by the patient and the HCP. The HCP issues the patient a home certificate to register with a home server. The patient and the respective home server generate the foreign certificates to access the foreign servers. The home server is responsible for linking the patient's data which is scattered across foreign servers. The home server aggregates the data and delivers it to the HCP. The SPDC is scalable (E1) and applies the principle of load distribution (E2).

4.7 Discussion

The innovative SPDC framework is designed to facilitate data collection from a number of mobile patients in an efficient, secure, and ID-privacy preserving manner. This is achieved by the following. First, the SPDC allows the patient to upload his/her health data anonymously on any data collection servers. The patient selects one of the servers to be the home server, while the rest are considered as foreign servers. The foreign servers are those used to collect the patient's data. The home server is the one that is used to anonymously link the patient's data scattered across foreign servers into one account and then forward it to the HCP. Second, the patient uses a new index pseudonym for each data collection server; this pseudonym is like an account name. Third, for multiple requests with the same server (e.g. DCsrv_1), the patient generates a request pseudonym based on the index pseudonym used to identify the patient with DCsrv_1. This is to allow only the DCsrv_1 to link multiple requests for the same patient. Third, to allow only an authorized patient to use the data collection service, SPDC uses a digital certificate. The HCP is responsible for generating a home certificate for the patient to register with any server as a home server. The patient generates the foreign certificates which are blindly signed by the home server to allow the patient to access foreign servers. Fourth, SPDC is patient-centric as the patient is involved in selecting the service providers, generating the pseudonyms that will be used with other service providers (data collection servers), and generating the pseudonyms credentials to access foreign servers. Fifth, SPDC provides the patient with a priority-based service without any server (except the authorized healthcare provider) being able to analyze the health data for the patient. To implement the SPDC framework, Chapter 5 presents the methods used to generate and link types of pseudonyms, Chapter 6 presents the anonymous distributed authentication method and protocols, and Chapter 7 shows the methods and protocols used to upload authenticated data on multiple foreign servers and forward this data to the home server and then to the HCP.

4.8 Chapter Summary

In this chapter, we have explained the SPDC architecture, its entities, and the interaction among the entities of the architecture. We started by proposing the first data collection (FDC) system. Then, in the FDC system, we have identified a number of security and performance issues. To tackle these issues, we have proposed the SPDC framework. We have explained the SPDC architecture, its entities, components on each entity, and how they should interact with each other. The next chapters will present the methods used to design the SPDC framework. These methods include a method for generating and linking pseudonyms, and a method for generating pseudonym certificates. Then, we will present secure protocols to show how to perform anonymous authentication and uploading.

Chapter 5

Novel Pseudonym Generation and Linkage Methods

5.1 Chapter Introduction

This chapter focuses on how to generate pseudonyms to hide a patient's real ID and to link these pseudonyms back to the patient's real ID. The patient uses a pseudonym ID called an index pseudonym for each server and one session ID for each request with the same server. The purpose of generating different pseudonyms for different servers and sessions is to prevent tracing the patient across different servers and sessions. The purpose of the linking method is to allow only an authorized entity to link the patient's session pseudonyms back to the index pseudonym and the index pseudonym back to the patient's real ID. This chapter starts by giving a survey of existing pseudonym generation methods. Then it describes two different methods that we have designed to generate and link pseudonyms. The first method is the Hierarchical Pseudonym Generation and Linkage (HPGL) method and the second is the non-Hierarchical Pseudonym Generation and Linkage (nHPGL) method. The nHPGL method offers a better performance than the HPGL method; it does not require searching for the verification key in the server (verifier) database when verifying pseudonyms, thus shortening pseudonym verification time and improving efficiency.

In detail, the chapter is organized as follows. Section 5.2 provides a literature survey of pseudonym generation methods in health and general systems. Section 5.3 presents the design preliminaries (requirements, novel ideas, notation, assumptions, attack model) for designing the methods. Section 5.4 describes the detailed design of the HPGL method. Section 5.5 describes the detailed design of the nHPGL method. Section 5.6 analyses both methods in terms of requirements, performance, degree of anonymity and security. Finally, section 5.7 provides a summary of the chapter.

5.2 Patient ID Privacy: A Literature Survey

This section provides a survey of existing pseudonym generation methods proposed for the healthcare context and for non-healthcare (or general) context. Then, it highlights the limitations in applying these methods directly.

5.2.1 Pseudonym Generation in Health Systems

Researchers [41]–[43] have suggested solutions to preserve the privacy of a patient's ID across multiple healthcare providers. The main idea behind the solutions is that each patient should have multiple pseudonyms, one for each healthcare provider. They suggest that these pseudonyms should be linked to the patient's real ID without the need for searching a mapping table. To generate linkable pseudonyms they suggest using an encryption technique, and a decryption technique to recover the real ID. The encryption and decryption mechanisms can be symmetrical or asymmetrical. In the symmetrical mechanism, the generation and linkage of the pseudonyms is done by using the same secret key. To generate a pseudonym, the patient's ID is encrypted using the secret key. To link the pseudonym back to the patient's ID, it is decrypted using the same secret key. In the asymmetrical mechanism, the generation and linkage of the pseudonym are done by using a key pair of a public and a private one. The asymmetrical mechanism is used to generate linkable pseudonyms if we want to strictly control who can link pseudonyms back to the patient's ID. The drawback in these solutions is that the generated pseudonyms are static (i.e., they are used for a long time).

A method [41] proposed by Zhang et al suggested using a cryptographic algorithm to map different pseudonyms of a single patient to a unique pseudonym known as a HealthGridID pseudonym. The solution consists of two functions namely the NHSNo-to-Pseudonym (N2P) conversion module and Pseudonym-to-HealthGridID (P2H) conversion module. The first module can be run by an independent Trusted Third Party (TTP). The role of this module is to re-encode a patient's NHS number into a pseudonym. The second module's role is to convert each pseudonym (generated by N2P) into a unique pseudonym known as HealthGridID. The solution can be explained in more detail as follows. Suppose a patient (Alice) has different records with different healthcare providers. To preserve Alice's ID privacy, the system issues Alice multiple pseudonyms, one for each healthcare provider. When one healthcare provider uploads Alice's records into the HealthGrid, the system automatically converts Alice's pseudonym (known to the healthcare provider) into Alice's HealthGridID and stores Alice's records under a HealthGridID ID. This solution allows an external TTP to know the patient's NHS number and the patient's pseudonyms used with each healthcare provider.

The pseudonymization scheme [42], as seen in Figure 5.1, allows each healthcare provider to generate a local pseudonym for each patient using a hash function. These different local pseudonyms can be linked back to a global pseudonym ID by the TTP. However, the TTP will be unable to know the patient's NHS number. In more detail, the health distributed system first agrees upon public cryptographic parameters. Then each healthcare provider computes its own public and private keys, keeps the private key secret, and sends the public key to the TTP. Then each healthcare provider generates a local pseudonym by using a hash function and a private key then sends the local pseudonym to the TTP. The TTP calculates the global pseudonym ID for the patient based on the different pseudonyms received from different healthcare providers. To link each pseudonym to the patient's real ID, each healthcare provider uses a mapping table to map the pseudonym to the real ID of the patient. The reason for using a mapping table is that the healthcare provider uses a hash function, which is a one way function. There is no way to use a reverse function to recover the real ID of the patient from a hash string.

To solve the problem with the use of the mapping table, we use the 3LI2Pv2 method proposed by Addas and Zhang [43]. This method uses an encryption method to generate a pseudonym and a decryption method to recover the patient's real ID from the pseudonym. In this method the authors suggested generating the pseudonym in a hierarchical way. In the 3LI2Pv2 method, the pseudonym has three-levels. The first level (L1) is generated by a central trusted third party (C-TTP) to hide a patient's real identity. The L1 pseudonym is used as a common identifier to index a particular patient's data stored in multiple healthcare providers. The second level (L2) is generated based on the first pseudonym (L1) and it is used to hide L1 from unauthorized entities. It is generated by the C-TTP, one for each healthcare provider. The L2 pseudonym is used by the healthcare provider to link multiple data records for the same patient. The third level (L3) is built on both pseudonyms (L1 and L2). The L3 pseudonyms are given to authorized users (e.g. physicians) to anonymously access and link the patient's records across multiple healthcare providers. The hierarchical pseudonyms method will now be described in more detail. The C-TTP generates a symmetrical key (e.g. AES) and then encrypts the real ID of the patient using the key. This results in the first pseudonym (L1). The C-TTP is the only entity that can reverse L1 to the real identity of the patient using the decryption mechanism using the same symmetrical key. To generate the second pseudonym (L2), the C-TTP first generates an asymmetrical key pair for each healthcare provider. It then encrypts the L1 pseudonym with the public key of the healthcare provider (HCP1). This results in the L2 for the HCP1. The C-TTP sends the result to the HCP1. Only the C-TTP can recover the real ID of the patient from the L2 pseudonym. Third pseudonyms (L3) are generated to be used as access credentials. These pseudonyms can be generated by both C-TTP and each healthcare provider. This method supports three levels of patient identity privacy preservation when accessing distributed patient records. The method has several notable features: (1) it uses credentials to support the three levels of access; (2) it simplifies key management distribution; (3) it allows for separation of duties among trusted third parties; and (4) it improves scalability.

5.2.2 Pseudonym Generation in General Systems

The first system which used pseudonyms to hide the relationship between user transactions and their real identities in communicating with multiple organizations was proposed by Chaum [44]. In this proposed system, each user has a number of pseudonyms generated by a trusted third party (TTP), one pseudonym for each organization a user wishes to communicate with. These pseudonyms are not linkable (i.e., it is quite difficult for organization A and organization B to collude to link pseudonyms to the user's real identity). However, the proposed system relies heavily on a trusted third party (TTP) to generate pseudonyms. In addition, these pseudonyms are used to identify the user to the organization. When the user wants to access the same organization multiple times, they use the same pseudonym. This may help

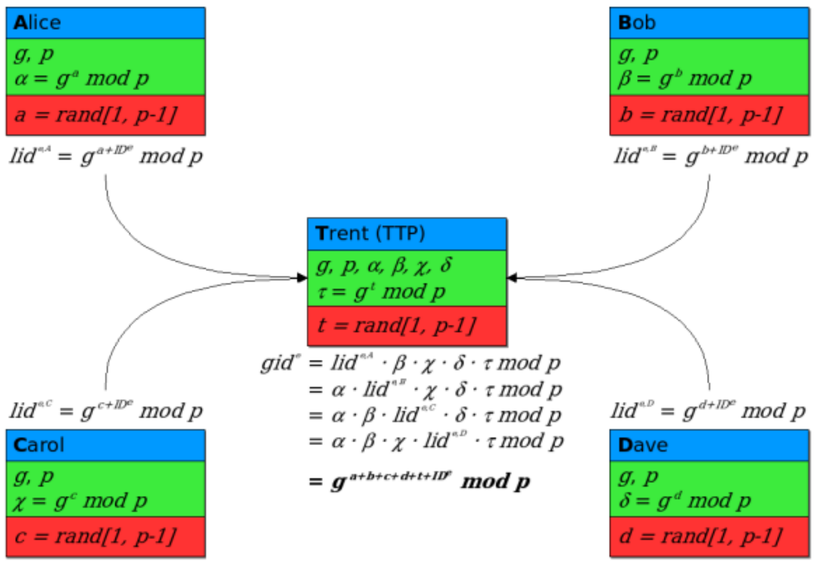

Alice
g, p
$\alpha = g^a \bmod p$
$a = rand[1, p-1]$
$lid^{e,A} = g^{a+ID^e} \bmod p$
Bob
g, p
$\beta = g^b \bmod p$
$b = rand[1, p-1]$
$lid^{e,B} = g^{b+ID^e} \bmod p$
Trent (TTP)
$g, p, \alpha, \beta, \chi, \delta$
$\tau = g^t \bmod p$
$t = rand[1, p-1]$
$gid^e = lid^{e,A} \cdot \beta \cdot \chi \cdot \delta \cdot \tau \bmod p$
$= \alpha \cdot lid^{e,B} \cdot \chi \cdot \delta \cdot \tau \bmod p$
$= \alpha \cdot \beta \cdot lid^{e,C} \cdot \delta \cdot \tau \bmod p$
$= \alpha \cdot \beta \cdot \chi \cdot lid^{e,D} \cdot \tau \bmod p$
$= g^{a+b+c+d+t+ID^e} \bmod p$
$lid^{e,C} = g^{c+ID^e} \bmod p$
Carol
g, p
$\chi = g^c \bmod p$
$c = rand[1, p-1]$
$lid^{e,D} = g^{d+ID^e} \bmod p$
Dave
g, p
$\delta = g^d \bmod p$
$d = rand[1, p-1]$

Figure 5.1: The pseudonymization scheme

an adversary to link multiple accesses between that user and that organization. One solution is to allow the user to generate a fresh pseudonym for each access.

Lysyanskaya proposed a pseudonym scheme [45] to decrease dependency on a TTP and to allow the user to generate pseudonyms. In this scheme, each user chooses a secret key independently. The secret key is only known to the user. The pseudonym is generated in cooperation between the user using his/her secret key and the organization using its secret key. No TTP is involved in the system. Instead, the organization and the user cooperate to generate and authenticate the pseudonym using a discrete logarithmic algorithm.

To allow the user to take control of generating the pseudonyms used by service providers and link these pseudonyms back to the user's real identity, Camenisch [46] proposed a new pseudonym system. This system allows a user to generate pseudonyms and to be able to link pseudonyms to his/her real identity. To do so, a blind converter (X) is introduced between the user and each organization. To generate the pseudonym for each organization, the user blindly and jointly computes the pseudonym for an organization (A) using the blind converter (X) and the user's real identity. Suppose the pseudonym for the organization A is $\mathrm{PsIDA} = \mathrm{UID}^{\mathrm{xa}} \pmod p$ and the pseudonym for organization B is $\mathrm{PsIDB} = \mathrm{UID}^{\mathrm{xb}} \pmod p$, where p is a prime number and xa, xb are public keys for system A and B respectively. Because of the security of the discrete log neither organization A nor organization B will be able to learn the value of the user real ID. Suppose now the service provider A would like to enquire about the user identified at A as PsIDA, from another service provider (B). Since A does not know the user's pseudonym at B, A encrypts the PsIDA under B's public key (i.e., $C = \mathrm{Enc}(\mathrm{pubB}, \mathrm{PsIDA})$). Then A sends the request to the blind converter (X) which uses homomorphic encryption to raise this cipher to the power of (xb/xa, where xa and xb are public keys for organization A and B respectively). This becomes like ($\mathrm{C} = \mathrm{Enc}(\mathrm{pubB}, \mathrm{PsIDA})^{\frac{\mathrm{xb}}{\mathrm{xa}}}$). The converter then passes this cipher value to system B, and through the homomorphic encryption can drive ($\mathrm{C} = \mathrm{Enc}(\mathrm{pubB}, \mathrm{UID}^{\mathrm{xb}})$). Organization B can then use its private key to get the pseudonym at B ($\mathrm{UID}^{\mathrm{xb}}$). Now, organization B will know the pseudo ID, but the blind converter or organization A cannot determine the ID stored at organization B.

5.2.3 Limitations in Existing Methods

There are limitations when applying the existing methods to satisfy our design requirements. These limitations are as follows.

- The first limitation is that these pseudonyms generated by the solutions are used as main pseudonyms. This means that they are used to identify the patient to healthcare providers or in general to service providers, with one pseudonym for each service provider. In a situation where the patient needs to access the service provider multiple times (e.g. for data uploading), by using the main pseudonym multiple times, an adversary may link multiple accesses by the same patient with the service provider.
- The second limitation is that some of these solutions rely on a Trusted Third Party (TTP) to generate the main pseudonyms and link them back. This (i.e., using the TTP) would decrease the level of ID privacy as the TTP would be able to learn each pseudonym a patient is using with each service provider or the TTP may be compromised by a skilled attacker. As a result, the private information of all patients is threatened. In addition, using one entity (e.g. using a single TTP) to generate and link each pseudonym would impose a performance bottleneck on the system.
- The third limitation is that the main pseudonym is fixed for each service provider. This would allow revealing the real ID of the patient over some time by using cryptographic analysis tools or by linking the patient's information from an external third party.

5.3 Design Preliminaries

This section presents the notation, assumptions and requirements used in the design of pseudonym generation and linkage methods.

5.3.1 Design Requirements

The following requirements are specified for the design of the pseudonym generation and linkage methods.

R1. Confidentiality: The pseudonym should hide the real identity of the patient.

R2. Conditional unlinkability: An unauthorized entity should not be able to link pseudonyms for the same patient across multiple servers or across multiple sessions (requests) with the same server.

R3. Uniqueness: The pseudonym should be unique. This means that no two patients should have the same pseudonym at any given time.

R4. Short life span: The pseudonym should have a short life span.

R5. Efficiency: The size of the pseudonym, its computation (generation) time, and its verification time should be as small as possible.

R6. Separation of duties: The generation and linkage of pseudonyms should be scattered among entities.

R7. Patient-centric: The patient should involve in the generation of pseudonyms. The pseudonyms variation will not be affected as we use a random text.

5.3.2 High-Level Idea

To satisfy the above requirements we have the following ideas.

Idea 1: Use multiple server multiple session pseudonyms. For each patient, there are two types of pseudonym: an index pseudonym and a request pseudonym. The index pseudonym is used as a unique artificial ID in place of the patient's real ID to identify the patient with each data collection server. The request pseudonym is used to identify each session with the same data collection server. The request pseudonym is generated based on the index pseudonym. There are two forms of index pseudonym: a Home Index Pseudonym (HIP) and Foreign Index pseudonym (FIP). The HIP is used to identify the patient with the home server. The FIP is used to identify the patient with foreign servers, one FIP for each foreign server. Regarding the request pseudonym, it has two forms: Home Request Pseudonyms (HRPs) and Foreign Request Pseudonyms (FRPs). The HRPs are generated based on the Home Index Pseudonym (HIP), while the FRPs are generated based on an FIP. The HIP is generated based on the real ID of the patient. The FIP is generated based on the home index pseudonym. The home request pseudonym is generated based on the home index pseudonym. The FRP is generated based on the foreign index pseudonym. This idea aims to satisfy the requirements R1 and R2.

Idea 2: Separate the functions of pseudonym generation and linkage among entities, as seen in Table 5.1. The HCP and patient are responsible for the generation of the index pseudonyms. The HCP generates the HIP based on the patient's real ID. The patient is responsible for the generation of the FIP based on the HIP. The patient is also responsible for the generation of the request pseudonyms the home and the foreign one. The generation of the request pseudonyms is on-demand (i.e., when the patient needs to communicate with a home or foreign server). The linkage function is done by the HCP, the home, and foreign servers. The HCP links the HIP to the real ID of the patient. This is because the HCP is the final destination for the patient data. Therefore, the HCP must be able to link the home index pseudonym to the patient real ID to be able to store the patient's data under the patient's real ID. The home server is the server responsible for linking the FIP to the HIP. This is because the home server will collect

Table 5.1: Entities that generate and link pseudonyms

Pseudonym	Generated by	Linked by	Known to	ID for
HIP	HCP or patient	HCP	Only HCP, home, patient	Home server
FIP	Patient	Home	Only patient and foreign	Foreign server
HRP	Patient	Home	Only patient and home	A home session
FRP	Patient	Foreign	Only patient and foreign	A foreign session

the patient's data from the foreign servers. Therefore, the collected data should be stored under the patient's HIP. Additionally, both the foreign and the home servers are responsible for linking each request pseudonym back to the index pseudonym. The home server links each home request pseudonym back to the HIP. The foreign server links each foreign request pseudonym back to the FIP. This is to store the patient's data under his/her account in the foreign server. This idea is to satisfy requirements R6 and R7.

Idea 3: To meet the efficiency requirements and prevent a single entity from linking all the patient's data scattered across servers, we design local and global linkability. The local linkability is done by the patient's home server and it links all the patient's data across servers to the patient's home index pseudonym. The global linkability links patient data to the patient's real ID and is done by the HCP.

Idea 4: To design an efficient request pseudonym verification method (to satisfy R5), we have two methods: one uses a tag number to specify the location of the verification key and the other one uses a non-hierarchical pseudonym, which facilitates finding the verification key without searching in the database.

Idea 5: The patient generates both the index and request pseudonyms. This supports the R3, R4, and R7 requirements. The generated pseudonyms are unique. This is because the key used to generate pseudonyms is different among patients.

5.3.3 Notation

The notation used throughout the method is summarized below.

HCP: The healthcare provider

$\mathrm{PID_i}$: Patient Real ID for patient i.

$\mathrm{MIP_i}$: Main Index Pseudonym for patient i.

$\mathrm{HIP_i}$: Home Index Pseudonym for patient i.

$\mathrm{FIP_i^f}$: Foreign Index Pseudonym for patient i with a foreign server f.

$\mathrm{HRP_{(i,r)}}$: Home Request Pseudonym for a patient i for a request r.

$\mathrm{FRP^f_{(i,r)}}$: Foreign Request Pseudonym for a patient i with a foreign server f.

$\mathrm{nHIP_i}$: Non-Hierarchical Home Index Pseudonym for patient i.

$\mathrm{nFIP_i^f}$: Non-Hierarchical Foreign Index Pseudonym for patient i with a foreign server f.

$\mathrm{nHRP_{(i,r)}}$: Non-Hierarchical Home Request Pseudonym for a patient i for a request r.

$\mathrm{nFRP^f_{(i,r)}}$: Non-Hierarchical Foreign Request Pseudonym for a patient i with a foreign server f.

$\mathrm{RPK_e}$: The RSA Public Key for entity e.

$\mathrm{RPR_e}$: The RSA Private Key for entity e.

$\mathrm{EPK_e}$: The Elliptical Curve Public Key for entity e.

$\mathrm{EPR_e}$: The Elliptical Curve Private Key for entity e.

$\mathrm{SK^1_{HCP}}$: The HCP First Secret Key.

$\mathrm{SK^2_{HCP}}$: The HCP Second Secret Key.

$\mathrm{SK_i}$: The Shared Key between patient i and HCP.

$\mathrm{HSK_i}$: The Home Shared Key for patient i with a home server.

$\mathrm{FSK_i^f}$: The Foreign Shared Key for patient i with a foreign server f.

$\mathrm{tPK_i^f}$,$\mathrm{tPR_i^f}$: Temporal Public and Private Keys generated by patient i with a foreign server f.

Enc, Dec: Asymmetrical Encryption and Decryption.

E, D: Symmetrical Encryption and Decryption.

T : Time stamp.

Inc: Index cryptographic nonce.

Rnd : Random value.

5.3.4 Assumptions

The following assumptions are used:

A1. The communication between the patient and any data collection server takes place over the Internet using an anonymous communication channel.

A2. The request message (e.g. uploading request) that a patient sends to a data collection server has the following fields, as shown in Figure 5.2. The message ID, the patient's request pseudonym which can be a hierarchical or non-hierarchical request pseudonym, the ID of the data collection server (Server-ID), the tag number (TagNo) (used only when the request pseudonym is a hierarchical), the patient's data, and the Message Authenticated Code (MAC) generated using the patient's shared key with the server.

A3. The key that is used by the patient to generate the request pseudonym is generated by using the Diffie–Hellman key exchange (EKeyExg).

A4. The index nonce was sent to the patient by the respective server.

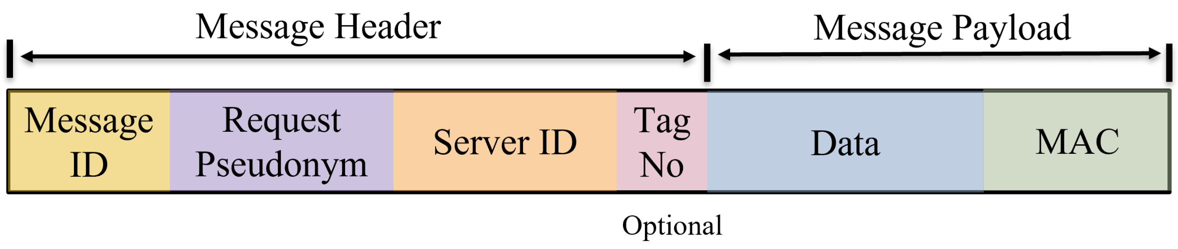


Figure 5.2: A request message format

5.3.5 The Attack Model

This section describes possible security threats that may be encountered during interactions between entities in the SPDC system. There are two types of attacks: external and internal attacks. External attacks occur when entities who have not been granted any privileges try to abuse the privileges of authorised entities to gain access to patient data. Internal attacks occur when authorised entities who have been granted some privileges try to abuse and elevate their privileges to gain access to data that is not covered by these privileges.

- Impersonation: is where an unauthorized entity pretends to be authorized (e.g. a patient, a data collection server) to gain access to a resource and patient privacy information. Impersonation usually occurs when an attacker successfully steals the credentials of an authorized entity.

- Pseudonym forgery/theft: occurs when an unauthorized entity generates a valid patient pseudonym, intending to deceive a data collection server and misuse the pseudonym to access sensitive patient data. The unauthorized entity could also obtain a valid pseudonym by stealing it from a legitimate entity, such as a trusted party or a patient.

- Brute force attack: is a strategy that involves checking all possible binary combinations of a secret (e.g. a key) until the correct one is found. A dictionary attack is a kind of brute force attack. Unlike a brute force attack, where a large proportion of key space is searched systematically, a dictionary attack tries only the possibilities that are most likely to succeed.

- Replay attack: is where an unauthorized entity records a transaction message and reuses it later to do something malicious (e.g. get access to patient data or any other confidential data).

- Repudiation attack: is where an entity takes part in a transaction then denies involvement in the transaction. For example, when the patient performs an action like generating a pseudonym, then they deny performing that action.

- Linkability attack: is where an unauthorized entity can link multiple actions or pseudonyms for the same patient. These attacks can happen for the following reasons. The periodic nature of the patient's data means that the patient must upload his/her data on the same server multiple times in one session (assuming that one session is 30 minutes long). If the patient is using the same ID for each upload, this might help the attacker to learn that these IDs are for one patient. Then by using a crypto-analysis tool, the unauthorized entity can guess the secret key used to generate these IDs. If the patient uses a tag to distinguish urgent data from normal data, an attacker may learn even more about the patient's health status. In addition, an authorized entity or an unauthorized one can infer that multiple IDs are for the same patient. This is done by observing the communication pattern.

5.4 Hierarchical Pseudonym Generation and Linkage Method

This section provides a detailed design for types of hierarchical pseudonyms, algorithms for generating, linking, and verifying the pseudonyms. It is worth mentioning that an index pseudonym is verified using a digital signature and certificate called the pseudonym certificate. The detailed design of this certificate and how to generate it will be discussed in Chapter 6. In this chapter, we only show how to verify the request pseudonyms using the tag method.

5.4.1 Pseudonym Types

There are four types of hierarchical pseudonyms as follows. Figure 5.5 shows the structure of hierarchical pseudonyms.

- Home Index Pseudonym (HIP): Each patient has a single home index pseudonym that is generated by the healthcare provider server (HCP). The HCP generates the home index pseudonym based on the Main Index Pseudonym (MIP). The MIP is generated based on the real ID of the patient. The purpose of the HIP is (1) to increase the level of protection by adding another layer on the MIP, (2) to be used as an ID for the patient with the home server. The HIP is only known to the HCP, the home server, and the patient. The HIP can only be linked back to the real ID of the patient by the HCP. It should be noted that the HIP consists of the MIP and the real ID of the patient.

- Foreign Index Pseudonyms (FIPs): Each patient has many FIP, one for each foreign server. These FIPs are generated by the patient (i.e., mobile device) based on the patient's Home

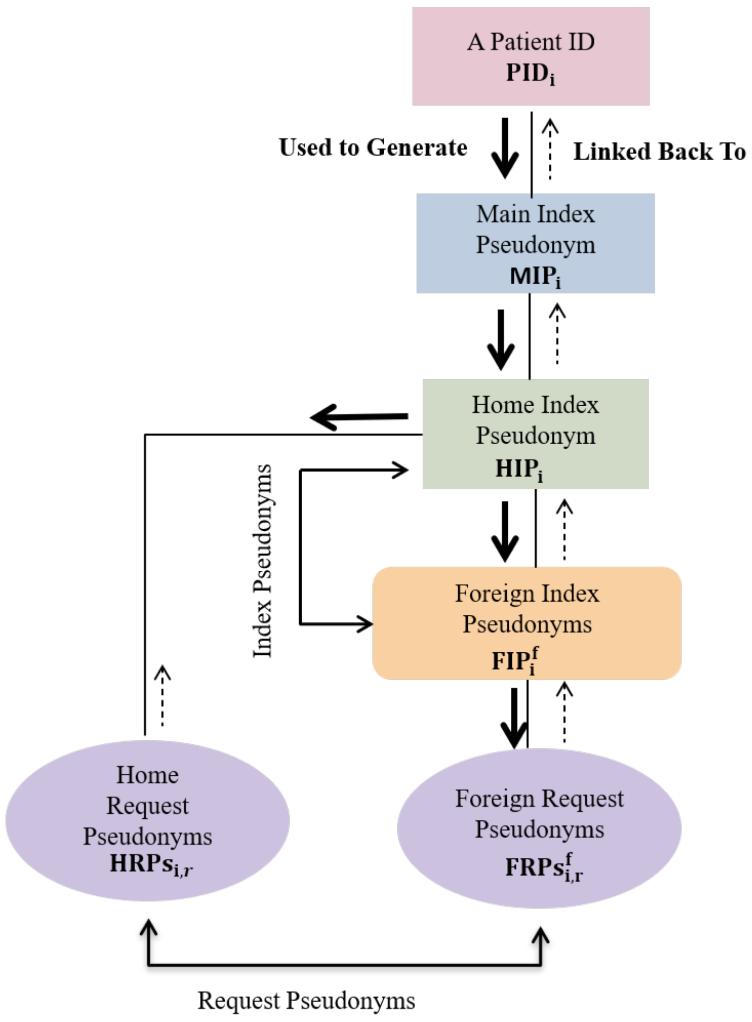


Figure 5.3: Hierarchical pseudonyms.

Index Pseudonym (HIP). The purposes of the FIP are to (1) hide the HIP of the patient (as no server is allowed to learn the index pseudonym used by the patient with another data collection server), (2) to be used as an ID for the patient with a foreign server. The purpose of having different FIPs for each foreign server is to break the link among different accesses by the same patient to different data collection servers. As explained previously, all the FIPs can be linked back by the home server to the HIP of the patient. This is because the home server is the anonymous linkage to the patient's data stored across different foreign servers. It should be noted that the home server does not have any previous knowledge of how the patient will generate each FIP. However, it still can link each FIP back to the patient's HIP (as explained in the algorithm section).

- Home Request Pseudonyms (HRPs): Each patient has a number of home pseudonyms. These pseudonyms are used to identify each request carried out by the patient with the home server. These pseudonyms are generated by the patient based on the patient's HIP. Only the home server is able to link these home request pseudonyms back to their origin (i.e, home index pseudonym). The purpose of using home request pseudonyms is to break the link between different requests carried out by the same patient with the home server. This would prevent an unauthorized entity from linking these request pseudonyms to a single patient and working out his/her real ID.

- Foreign Request Pseudonyms (FRPs): Each patient has a number of foreign request pseudonyms. These pseudonyms are used to identify each data upload carried out by the patient with a foreign server. They are generated by the patient based on a patient's foreign index pseudonym. Only the foreign server which knows the FIP is able to link these FRPs back to their origin (i.e., foreign index pseudonym). The purpose of FRPs is to break the link between different data uploads carried out by the same patient with the foreign server. This would prevent an unauthorized entity from linking these pseudonyms to a single patient and working out his/her real ID.

5.4.2 Hierarchical Pseudonym Verification (HPVer)

This method allows the server to limit the search for a patient's verification key (shared key). This shared key is used to generate a request pseudonym and generate a MAC on the request message. The same key is used to link the request pseudonym back to the index pseudonym. Therefore, this method enables the server to limit the search for a patient's verification key between two boundaries which we call the min and max boundary. Without this method, each server will try all verification keys to verify each request pseudonym.

This method can be explained as follows. First, the server should partition the database into a number of similar segments, as seen in Figure 5.4. Each segment contains a specific number of rows (say 10 rows). The first row of the segment we call the min boundary. The last row of the segment we call the max boundary. Suppose that each segment contains 10 rows, then the min and max boundaries of the first segment are 0 and 9 respectively. By using Algorithm 1, the server can calculate the min and max boundaries for each patient.

When the patient registers with the data collection server for the first time, the server stores the patient's information in one of the segments. Then, the server sends the patient the min and max boundaries of the segment. Suppose the server stores the patient's information in the first segment, then the min and max boundaries are 0 and 9 respectively. This means that the patient can use any number from 0 to 9 as a tag for each request. Suppose now the patient sends a request to the data collection server, and the request message contains a fresh request pseudonym, a tag number (based on our example the tag should be any number from 0 to 9), and the MAC generated using the shared key (verification key). The server receives the request but needs to authenticate the patient first. To authenticate the patient the server needs to find the verification key. To know in which segment to search for the verification key, the server uses the tag number as an input to Algorithm 2. This algorithm is used to re-calculate the min and max boundaries first. Once it has found the min and max boundaries, the server learns in which segment to search for the verification key. The server will try all the verification keys which are stored in the first segment (based on the example 0 to 9) to authenticate the patient. The authentication is done using MAC verification. If no match is found, the server rejects the request.

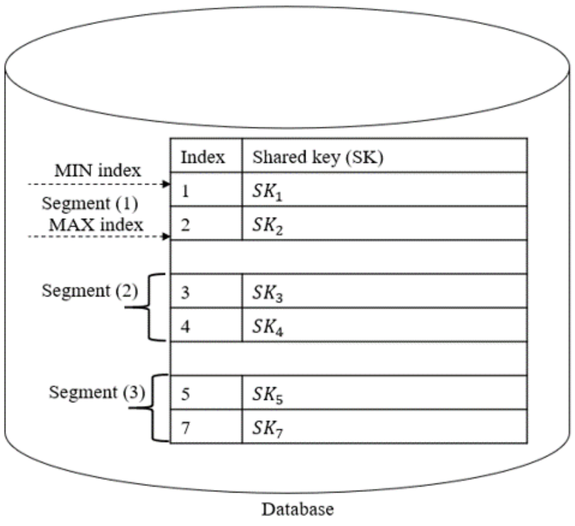


Figure 5.4: Database partitioning

Algorithm 1: Calculate Boundaries

```
Input  : Indx, Base
Output : Min, Max
Num = Indx % Base ;
Min = Indx - Num;
Max = Min + (Base-1);
```

5.4.3 Pseudonym Generation and Linkage Algorithms

This section provides details of hierarchical pseudonyms and their generation and linkage algorithms. The algorithms are as follows: the Home Index Pseudonym Generation (HIP-Gen) and Linkage (HIP-LnK) algorithms, Foreign Index Pseudonyms Generation (FIPs-Gen) and Linkage (FIPs-Lnk) algorithms, Home Request Pseudonyms Generation (HRPs-Gen) and Linkage (HRPs-LnK) algorithms, and Foreign Request Pseudonyms Generation (FRPs-Gen) and Linkage (FRPs-LnK) algorithms. As the names suggest, the generation algorithms are used to generate pseudonyms. The linkage algorithms are used to link the pseudonyms back to the index pseudonym. It is worth mentioning that, the shared key that is used to generate

Algorithm 2: Find Verification Key

```
Input  : Tag, Base, Msg, MAC1
Output : SK
Num = Tag % Base;
Min = Tag - Num;
Max = Min + (Base-1);
for (i=Min, i<=Max, i++)
{
SK = GetKey(i);   /* Get a Key from the DB*/
MAC = HMAC (Sk, Msg);  /* MAC Generation */
if (MAC==MAC1)  /* Comparing MACs */
return SK;
else
continue;
}
```

the MAC on a request message is the same as that which is used to generate the request pseudonym.

5.4.3.1 The HIP-Gen Algorithm

The HIP-Gen is executed by the HCP using two symmetrical keys (SK^1_{HCP}, SK^2_{HCP}) that are generated by and known only to the HCP. The HIP-Gen algorithm is shown in Algorithm 3, where E is a symmetric encryption function, such as AES. The HIP-Gen takes four inputs, encrypts them, and produces the home index pseudonym. The inputs to HIP-Gen are two symmetrical keys (SK^1_{HCP} and SK^2_{HCP}), a patient's real ID (PID_i), the current time (T), and a random number (Rnd). The algorithm for generating a home index pseudonym works as follows. In the first step, the HCP encrypts the patient's real ID (PID_i) with the current time (T) using the first key (SK^1_{HCP}). The output of this process is an index pseudonym called the main index pseudonym (MIP_i). This pseudonym is known only to the HCP. In the second step, the HCP encrypts the MIP_i with Rnd, using the second key (SK^2_{HCP}). This results in the home index pseudonym (HIP_i).

Algorithm 3: HIP-Gen Algorithm

```
Input  : $SK^1_{HCP}$, $SK^2_{HCP}$, $PID_i$, T, Rnd
Output $MIP_i$, $HIP_i$
:
1 $MIP_i = E(SK^1_{HCP}, PID_i \| T)$.
2 $HIP_i = E(SK^2_{HCP}, MIP_i \| Rnd)$.
```

5.4.3.2 The HIP-Lnk Algorithm

The HIP-Lnk algorithm is the reverse of the HIP-Gen process executed by the healthcare provider server (HCP). The HIP-Lnk algorithm is shown in Algorithm 4.

Algorithm 4: HIP-Lnk Algorithm

```
Input  : $HIP_i$, $SK^1_{HCP}$, $SK^2_{HCP}$
Output $PID_i$
:
1 $MIP_i \| Rnd = D(SK^2_{HCP}, HIP_i)$.
2 $PID_i \| T = D(SK^1_{HCP}, MIP_i)$.
```

5.4.3.3 The FIPs-Gen Algorithm

The FIPs-Gen is executed by a patient's mobile device to generate one FIP for each foreign server. This generation process is done by encrypting the HIP with other parameters. The encryption is done using the RSA public key of the home server as seen in Algorithm 5. The inputs to FIPs-Gen are the public key of home server (RPK_h), the HIP (HIP_i), a random number (Rnd), and the current time (T). The output from FIPs-Gen is a FIP (FIP^f_i). The reason for using the random number (Rnd) and the current time (T) to generate the FIP is to make it different each time and difficult to predict by an unauthorized entity.

Algorithm 5: FIPs-Gen Algorithm

```
Input  : $RPK_h$, $HIP_i$, T, Rnd
Output : $FIP_i^f$
1 $FIP_i^f$ = Enc ($RPK_h$, $HIP_i$ || T || Rnd)
```

5.4.3.4 The FIPs-Lnk Algorithm

The FIPs-Lnk algorithm is the reverse of the FIPs-Gen process executed by the home server of the patient.

Algorithm 6: FIPs-Lnk Algorithm

```
Input  : $RPR_h$, $FIP_i^f$
Output : $HIP_i$
1 $HIP_i$ || T || Rnd = Dec ($RPR_h$, $FIP_i^f$)
```

5.4.3.5 The HRPs-Gen Algorithm

The HRPs-Gen algorithm is executed by the patient's mobile device. The purpose of this algorithm is to generate a fresh pseudonym for each request going to the home server. The HRPs-Gen algorithm uses Algorithm 7. The HRPs-Gen algorithm takes four inputs, performs an encryption, and produces one output which is the home request pseudonym ($HRP_{(i,r)}$). The inputs to HRPs-Gen algorithm are the home index pseudonym (HIP_i), a home shared key (HSk_i), a random number (Rnd), and an index nonce (Inc). The home shared key is a symmetrical key generated using the EKeyExg algorithm. The purpose of using an index nonce (Inc) is to prevent an attacker from reusing the home request pseudonym to launch the replay attacks.

Algorithm 7: HRPs-Gen Algorithm

```
Input  : $HSk_i$, $HIP_i$, Rnd, Inc
Output : $HRP_{(i,r)}$
1 $HRP_{(i,r)}$ = E ($HSk_i$, $HIP_i$ || Rnd || Inc).
```

5.4.3.6 The HRPs-Lnk Algorithm

The HRPs-Lnk algorithm is the reverse of the HRPs-Gen process executed by a patient's home server. The purpose of HRPs-Lnk algorithm is to link each HRP to the corresponding HIP. The HRPs-Lnk algorithm uses Algorithm 8.

Algorithm 8: HRPs-Lnk Algorithm

```
Input  : $HSK_i$, $HRP_{(i,u)}$
Output  $HIP_i$
:
1 $HIP_i \| Rnd \| Inc = D(HSK_i, HRP_{(i,r)})$.
```

5.4.3.7 The FRPs-Gen Algorithm

The FRPs-Gen algorithm is executed by a patient's mobile device. The purpose of this algorithm is to generate a fresh pseudonym for each upload request. The FRPs-Gen algorithm uses Algorithm 9. The FRPs-Gen method takes five inputs, performs encryption, and produces one output, the FRP ($FRP^f_{(i,r)}$). The inputs to FRPs-Gen are a FIP (FIP^f_i), a foreign shared key (FSK^f_i), a random number (Rnd), a priority tag (PR), and an index nonce (Inc). The key (FSK^f_i) is a symmetrical key generated using the EKeyExg algorithm. The purpose of using the priority tag (PR) is to allow the foreign server to learn the health condition of the patient without the need to decrypt the patient's data.

Algorithm 9: FRPs-Gen Algorithm

```
Input  : $FSK^f_i$, $FIP^f_i$, PR, Rnd, Inc
Output  $FRP^f_{(i,r)}$
:
1 $FRP^f_{(i,r)} = E(FSK^f_i, FIP^f_i \| Pr \| Rnd \| Inc)$
```

5.4.3.8 The FRPs-Lnk Algorithm

The FRPs-Lnk algorithm is the reverse process of the FRPs-Gen executed by a foreign server. The purpose of FRPs-Lnk algorithm is to link each FRP to the corresponding FIP. The FRPs-Lnk algorithm uses Algorithm 10.

Algorithm 10: FRPs-Lnk Algorithm

```
Input  : $FSK^f_i$, $FRP^f_{(i,r)}$
Output  $FIP^f_i$
:
1 $FIP^f_i \| Pr \| Rnd \| Inc = D(FSK^f_i, FRP^f_{(i,r)})$
```

5.4.4 An Issue with The Hierarchical Method

Our aim is to build a verification method that can verify any request pseudonym without the need to search the database for the verification key. This aim can be satisfied by changing the way we generate the request pseudonyms and, consequently, the index pseudonyms. This leads us to the new method, the non-hierarchical method. The main idea of this method is to use elliptical curve public keys as pseudonyms. The following explains the idea in more detail.

- The HIP is generated by the patient using the EKeyGen algorithm. The patient first generates a key pair, public and private keys. The public key (EPK) is sent to the HCP. Then the EPK will be the patient's non-hierarchical HIP.

- The FIP is generated based on the non-hierarchical home index pseudonym (recall it is an elliptical curve public key). To generate this pseudonym, the patient uses the RSA public key of the home server to encrypt the HIP with a random string to generate the FIP. When the home server links the foreign index pseudonym to the home index pseudonym of the patient, the home server will use the RSA private key to decrypt the FIP and discover the home index pseudonym. To find the verification key without searching the database, the home server will multiply its elliptical curve private key with the HIP (recall it is an elliptical curve public key). The multiplication process results in the verification key, as seen in the EKeyGen algorithm in Chapter 4.

- The HRP is generated based on the non-hierarchical home index pseudonym (recall it is an elliptical curve public key). To generate these pseudonyms, the patient uses the RSA public key of the home server to encrypt the HIP with a random string. When the home server links a non-hierarchical HRP to the home index pseudonym of the patient, the home server will use its RSA private key to decrypt the HRP and discover the HIP. To find the verification key without searching the database, the home server will multiply its elliptical curve private key with the HIP (recall it is an elliptical curve public key). The multiplication process results in the verification key. The verification key will be used to verify the MAC.

- The FRP is generated based on a temporal elliptical curve public key generated by the patient. Then, it uses the RSA public key of the foreign server to encrypt the temporal elliptic curve public key with a random string. When the foreign server links a FRP to the FIP of the patient, the foreign server will use its RSA private key to decrypt the FRP and discover the temporal elliptic curve public key. To find the verification key without searching the database, the foreign server multiplies its elliptical curve private key with the temporal elliptic curve public key which results in the verification key.

5.5 Non-Hierarchical Pseudonym Generation and Linkage Method

This section provides details for non-hierarchical pseudonyms and their generation and linkage algorithms. This method aims to eliminate the need for the search in the database for the verification key.

5.5.1 Non-hierarchical Pseudonym Verification (nHPVer) Method

This method uses the elliptical curve public key as a pseudonym. By using this method, the shared key between two entities can be obtained on the fly. This is done by the two entities (a patient and a server) without the need to exchange any parameter values except for the

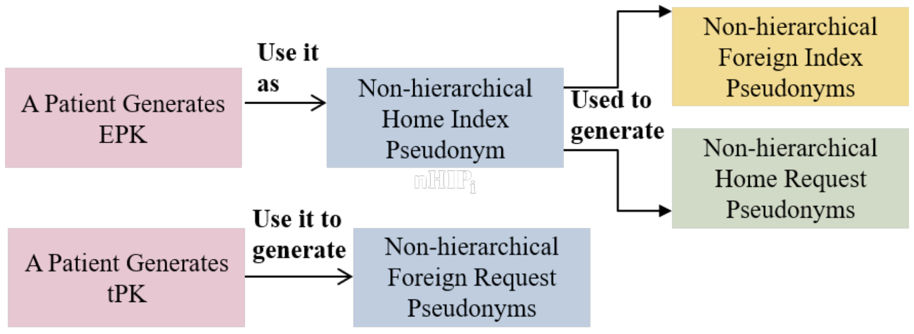

A Patient Generates EPK
Use it as
Non-hierarchical Home Index Pseudonym
nHIP_i
Used to generate
Non-hierarchical Foreign Index Pseudonyms
Non-hierarchical Home Request Pseudonyms
A Patient Generates tPK
Use it to generate
Non-hierarchical Foreign Request Pseudonyms

Figure 5.5: Non-hierarchical pseudonyms.

ones agreed upon during the system initialization phase. The server (a home server, foreign server, or HCP) calculates the shared key by multiplying the patient's elliptical curve public key with the server's elliptical curve private key. The patient calculates the shared key by multiplying his/her elliptical curve private key with the server's elliptical curve public key. This multiplication process results in the shared key.

5.5.2 Pseudonym Generation and Linkage Algorithms

This section provides details about non-hierarchical pseudonym generation and linkage algorithms that are used to generate and link each type of pseudonym. These algorithms are: non-hierarchical home index pseudonym generation (nHIP-Gen), non-hierarchical foreign index pseudonym generation (nFIPs-Gen) and linkage (nFIPs-Lnk) algorithms, non-hierarchical home request pseudonym generation (nHRPs-Gen) and linkage (nHRPs-LnK) algorithms, and non-hierarchical foreign request pseudonym generation (nFRPs-Gen) and linkage (nFRPs-LnK) algorithms.

5.5.2.1 The nHIP-Gen Algorithm

The Non-Hierarchical Home Index Pseudonym generation algorithm is executed by the patient. In this algorithm, the patient generates an elliptical curve public key (EPK) using the EKeyGen algorithm. This EPK will be the patient's non-hierarchical home index pseudonym (nHIP) with both the home server and healthcare provider (HCP).

5.5.2.2 The nHIP-Lnk Algorithm

The Non-Hierarchical Home Index Pseudonym linkage (nHIP-Lnk) algorithm is executed by the HCP. In this algorithm, the HCP links the non-hierarchical home index pseudonym (nHIP) to the patient's HIP.

5.5.2.3 The nFIPs-Gen Algorithm

The Non-Hierarchical Foreign-Index Pseudonyms Generation (nFIPs-Gen) algorithm is executed by a patient's mobile device using a RSA encryption and home server's RSA public key (RPK_h). The nFIPs-Gen algorithm uses Algorithm 11. The nFIPs-Gen takes four inputs, performs encryption, and produces the output. The inputs to nFIPs-Gen are the public key of the home server (RPK_h), the non-hierarchical home index pseudonym ($nHIP_i$), a random number (Rnd), and the current time (T). The output from the nFIPs-Gen is a non-hierarchical FIP ($nFIP_i^f$).

Algorithm 11: nFIPs-Gen Algorithm

```
Input  : $RPK_h$, $nHIP_i$, T, Rnd
Output : $nFIP_i^f$
1 $nFIP_i^f$ = Enc ($RPK_h$, $nHIP_i$ || T || Rnd).
```

5.5.2.4 The nFIPs-Lnk Algorithm

The Non-Hierarchical Foreign-Index Pseudonyms Linkage (nFIPs-Lnk) algorithm is the reverse process of nFIPs-Gen executed by a patient's home server. The nFIPs-Lnk uses Algorithm 12. To obtain the verification key (HSK_i), the home server multiplies its elliptical curve private key (EPK_h) by the patient's key EPK_i. Then, the home server uses the key to verify pseudonyms.

Algorithm 12: nFIPs-Lnk Algorithm

```
Input  : $RPR_h$, $nFIP_i^f$
Output : $nHIP_i$, $HSK_i$
1 $nHIP_i$ || T || Rnd = Dec ($RPR_h$,$nFIP_i^f$);
2 $HSK_i$ = $EPK_i$ * $EPK_h$. // Verification Key
```

5.5.2.5 The nHRPs-Gen Algorithm

The nHRPs-Gen is an algorithm executed by the patient's mobile device. The nHRPs-Gen uses Algorithm 13 to generate a fresh non-hierarchical HRP for each request carried out by the patient with his/her home server. The nHRPs-Gen algorithm takes five inputs, performs an asymmetrical encryption, and produces one output, the non-hierarchical HRP. The inputs to nHRPs-Gen are the non-hierarchical nHIP ($nHIP_i$), the asymmetrical public key of home server (RPK_h), an index nonce (Inc), and a random number (Rnd).

Algorithm 13: nHRPs-Gen Algorithm

```
Input  : $nHIP_i$, $RPK_h$, Rnd, Inc, Pr
Output : $nHRP_{i,r}$
1 $nHRP_{i,r}$= Enc($RPK_h$,$nHIP_i$||PR|| Rnd || Inc);
```

5.5.2.6 The nHRPs-Lnk Algorithm

The nHRPs-Lnk is the reverse process of nHRPs-Gen executed by the patient's home server to link each non-hierarchical HRP back to its non-hierarchical home index pseudonym (nHIP). The nHRPs-Lnk algorithm uses Algorithm 14.

```
Algorithm 14: nHRPs-Lnk Algorithm
Input  : $RPR_h$, $nHRP_{i,r}$
Output : $nHIP_i$
1 $nHIP_i$ || Rnd ||Inc = Dec ($RPR_h$,$nHRP_{i,r}$);
2 $HSK_i$ = $nHIP_i$ * $EPR_h$. // Verification Key
```

5.5.2.7 The nFRPs-Gen Algorithm

The nFRPs-Gen is an algorithm executed by the patient's mobile device to generate a fresh non-hierarchical foreign request pseudonym for each uploading request carried out by the patient with the foreign server. The nFRPs-Gen uses Algorithm 15. The nFRPs-Gen method takes five inputs, performs an asymmetrical encryption, and produces one output which is the non-hierarchical foreign request pseudonym. The inputs to nFRPs-Gen are a string format of the temporal public key ($stPK_i^f$), an asymmetrical public key of foreign server (RPK_f), index nonce (Inc), a random number (Rnd), and the priority tag (PR).

```
Algorithm 15: nFRPs-Gen Algorithm
Input  : $tPK_i^f$, $RPK_f$, Rnd, Inc, PR
Output : $nFRP_{i,r}^f$
1 $nFRP_{i,r}^f$= Enc($RPK_f$ , $tPK_i^f$||PR||Rnd||Inc)
```

5.5.2.8 The nFRPs-Lnk Algorithm

The nFRPs-Lnk is the reverse process of the nFRPs-Gen executed by the patient's foreign server. The nFRPs-Lnk method uses Algorithm 16.

```
Algorithm 16: nFRPs-Lnk Algorithm
Input  : $RPR_h$, $nFRP_{i,r}^f$
Output : $stPK_i^f$
1 $stPK_i^f$|| Pr || Rnd||Inc = Dec ($RPR_f$,$nFRP_{i,r}^f$)
2 $FSK_i^f$ = $tPK_i$ * $EPR_f$. // Verification Key
```

5.6 Analysis of Methods

The methods analysis includes requirements, security, and performance analysis.

5.6.1 Requirements Analysis

This section analyse the hierarchical and non-hierarchical methods against the design requirements.

R1. In the hierarchical method, the patient's real ID is protected under a number of layers of pseudonyms: The first layer is the main index pseudonym ($\mathrm{MIP_i}$) generated by the HCP using a symmetrical key ($\mathrm{SK^1_{HCP}}$). This key is known only to the HCP. Then, the HCP generates the home index pseudonym ($\mathrm{HIP_i}$) based on the main index pseudonym. This is done by encrypting the $\mathrm{MIP_i}$ using the second symmetrical key ($\mathrm{SK^2_{HCP}}$). This creates the second layer. The length of both keys is 128 bits. This requires $2^{128} * 2^{128}$ attempts to obtain the keys and discover the real ID of the patient. This would take to $1.02 * 10^{18}$ years as seen in Table 5.2. Moreover, when the mobile device of the patient generates the home request pseudonyms (HRPs) or foreign request pseudonyms (FRPs), it creates another layer to protect the patient's real ID. To work out the patient's real ID from his/her FIP involves guessing two symmetrical keys and one asymmetrical key, which requires $2^{128} * 2^{128} * 2^{2048}$ attempts. To work out the patient's real ID from the home request pseudonym (HRPs), involves guessing three symmetrical keys, which requires $2^{128} * 2^{128} * 2^{128}$ attempts. To work out the patient's real ID from the foreign request pseudonyms (FRPs), involves guessing three symmetrical keys and one asymmetrical key, which requires $2^{128} * 2^{128} * 2^{2048} * 2^{128}$ attempts. Regarding the non-hierarchical method, the patient's real ID is protected, as the patient uses the elliptical curve public key ($\mathrm{EPK_i}$) as a home index pseudonym.

R2. The pseudonyms can be linked by the authorized entity: In the hierarchical method, the home index pseudonym can be linked by the HCP to the patient's real ID. The foreign index pseudonyms for a particular patient can be linked back to the patient's home index pseudonym only by the home server selected by the patient. The foreign request pseudonyms are linked to the foreign index pseudonym by the foreign server which knows the patient from this foreign index pseudonym. The home request pseudonyms are linked back to the home index pseudonym only by the home server. Any unauthorized entity will not be able to link these pseudonyms to a particular patient for two reasons. The first is that to be able to link any pseudonym, the unauthorized entity needs the correct shared key. This key is very hard to obtain. The shared keys to link the patient's home index pseudonym to the patient real ID, are generated by the HCP. These keys are kept secret. The key used to link the foreign index pseudonyms to the patient's home index pseudonym is the home server's RSA private key. The key used by the home server to link the home request pseudonyms is called the home shared key (HSK). The key used by the foreign server to link the foreign request pseudonyms is called the foreign shared key (FSK). The keys (HSK and FSK) are generated using each party's elliptical curve private keys. Guessing these keys is like trying to solve the elliptic curve discrete logarithm problem (ECDLP), which has been proven over the years is not computationally feasible. The second reason is that we are using anonymous communication; this means that it is very hard for an unauthorized entity to correctly guess where this pseudonym comes from, the method which was used to generate the pseudonym (i.e., hierarchical or non-hierarchical), and the type of the pseudonym (i.e., index or request pseudonym). By using Shannon's entropy [47], Section 5.6 shows that linking multiple pseudonyms to a particular patient is a challenging task. In the non-hierarchical method, the home index pseudonym is an elliptical curve public key (EPK) generated by the patient and sent through a secure channel to the HCP. The HCP then learns the relation between the EPK and the patient's real ID. The non-hierarchical foreign index pseudonyms for a particular patient can be linked back to the patient's non-hierarchical home index pseudonym only by the home server selected by the patient. The foreign request pseudonyms are linked to a temporal elliptical curve public key by the foreign server which knows the relation between the non-hierarchical foreign index pseudonym and the temporal elliptical curve public in a certificate used in the authentication (we will explain the generation of the certificate in the next chapter). The non-hierarchical home request pseudonyms are linked back to the non-hierarchical home index pseudonym only by the home server. An unauthorized entity will not be able to link these pseudonyms to a particular patient for two reasons. The first is that to be able to link a pseudonym to a particular patient, one needs to know the relationship and the correct shared key. To link the patient's non-hierarchical home index pseudonym (which is EPK) to the patient real ID, one needs to know the relationship between them. This relationship is only known to the patient and the HCP. The key used to link the non-hierarchical foreign index pseudonyms to the patient's non-hierarchical home index pseudonym is the home server RSA private key. The key that is used by the home server to link the non-hierarchical home request pseudonyms is the home server's RSA private key. The key used by the foreign server to link the non-hierarchical foreign request pseudonyms is the foreign server's RSA private key. Obtaining the private key of an entity is a complex task, as explained in Chapter 3.

R3. Pseudonyms generated using both methods are unique: The proof of the uniqueness of the pseudonyms is based on the following facts. Fact 1: hierarchical pseudonyms are generated based on an NHS number, which is a unique ten digit number. This can ensure the generation of the pseudonyms different. Fact 2: in the generation process of each pseudonym, a random number is used as an input to the generation function.

R4. Pseudonyms generated using both methods have a short life: The HCP is responsible for generating a new hierarchical home index pseudonym, when the patient wants to change home server (i.e., every 24 hours). In the case of a non-hierarchical home index pseudonym, a new elliptical curve public key will be generated by the patient. In addition, in the case of hierarchical or non-hierarchical foreign index pseudonyms, the patient's mobile device generates a new hierarchical or non-hierarchical foreign index pseudonym for each foreign server. The foreign index pseudonyms are not fixed for each server, the patient generates a new pseudonym when s/he wants to use a previously used server. Moreover, for each request with a home or foreign server, the patient's mobile device generates a fresh hierarchical or non-hierarchical home or foreign request pseudonym.

R5. Efficiency: The verification time in the non-hierarchical method is more efficient than the hierarchical method. For example, when the home server receives the non-hierarchical home re-

Table 5.2: AES Key Size and Time to Crack

Key Size	Time to Crack
56-bit	399 seconds
128-bit	$1.02*10^{18}$ years
192-bit	$1.872*10^{37}$ years
256-bit	$3.31*10^{56}$ years

quest pseudonym, it decrypts the pseudonym using its private key to obtain the non-hierarchical home index pseudonym which is the elliptical curve public key (EPK_i). The home server multiplies its elliptical curve private key (EPR_h) with the patient's elliptical curve public key (EPK_i) to get the shared key and verify the message (i.e., $HSK_i = EPK_i * EPK_h$). The same is applied for a non-hierarchical foreign request pseudonym. The performance comparison between the two methods is presented in section 7.8.

R6. Both methods apply the principle of the separation of duties: In the hierarchical method, only the home index pseudonym is generated by the HCP, while the other pseudonyms are generated by the patient's mobile device. In the non-hierarchical method, all the pseudonyms are generated by the patient's mobile device. For both methods, the home index pseudonym is linked by the HCP, each foreign index pseudonym is linked by the patient's home server, each home request pseudonym is linked by the patient's home server, and each foreign request pseudonym is linked by the corresponding foreign server.

R7. Both methods are considered patient-centric as the patient is involved in pseudonym generation.

5.6.2 Threat Analysis

In this section, both methods are analyzed against security threats. Here, Alice is an authorized patient and Eve is an adversary.

Theorem 1. *Methods are protected against impersonation and pseudonym forgery attacks.*

Proof. In the hierarchical method, the generation of pseudonyms requires the knowledge of certain parameters that are hard to obtain by an unauthorized entity. To generate a home index pseudonym, the adversary should have knowledge of the two secret keys. To generate a foreign index pseudonym, the adversary should have knowledge of the public key of the home server of the patient, the home index pseudonym of the patient, the correct time, and the correct random number. As previously explained, the seed to the random generator is the hash value of the private key of the patient. The unauthorized entity may come to know the public key of the home server and guess other parameters, however, it is very hard for any entity to guess the random number. To generate a home or foreign request pseudonym, the unauthorized entity needs to know the following parameters: the home/foreign index pseudonym, an index nonce, the time, the random number, and the secret key (home or foreign shared key).

The secret keys are difficult to obtain because generating these keys requires the knowledge of the patient's private keys (i.e., EPK or tPK). Regarding the non-hierarchical method, the generation of pseudonyms requires the knowledge of certain parameters that are hard to obtain by unauthorized entities. To generate a non-hierarchical foreign index pseudonym, the unauthorized entity should know the following parameters: the public key of the home server of the patient, the non-hierarchical home index pseudonym of the patient, the correct time, and the correct random number. As explained previously, the seed to the random generator is the hash value of the private key of the patient. It is difficult to reproduce the same random number. To generate a non-hierarchical home/foreign request pseudonym, the unauthorized entity needs to know the following parameters: the non-hierarchical home index pseudonym (i.e., EPK) /temporal elliptical curve public key (tPK), an index nonce, the time, the random number, and the RSA public key of home/ foreign server. The random number is generated based on a secure random number generator using the hash value of the private key of the patient as a seed to generate a random number. It is hard to generate the same value for the home or foreign request pseudonym even under the assumption that the unauthorized entity could correctly guess the other parameters.

Theorem 2. *Methods are protected against replay attacks.*
Proof. In the hierarchical method, suppose Eve tries to replay a message that contains an old foreign request pseudonym to disturb the foreign server. After the foreign server receives the message, it tries to find the verification key (FSK_i^f) to verify the MAC of the message. When the server finds the verification key, it uses the key to decrypt the foreign index pseudonym (FIP_i^f) and discover the index nonce. The server then verifies the freshness of the index nonce; if it is an old nonce the server rejects the message. The same is done for the non-hierarchical method.

Theorem 3. *Methods are protected against brute force attacks.*
Proof. In the hierarchical method, suppose Eve wants to work out Alice's real ID from her home index pseudonym (HIP) using a brute force attack. This involves guessing two symmetrical keys, which requires up to $2^{128} * 2^{128}$ attempts to reveal Alice's real ID. To work out Alice's real ID from her foreign index pseudonym (FIP), Eve would need to guess two symmetrical keys and one asymmetrical key, which requires up to $2^{128} * 2^{128} * 2^{2048}$ attempts. To work out Alice's real ID from her home request pseudonym, Eve would need to guess three symmetrical keys, which requires up to $2^{128} * 2^{128} * 2^{128}$ attempts. To work out Alice's real ID from a foreign request pseudonym, Eve would need to guess three symmetrical keys and one asymmetrical key, which requires $2^{128} * 2^{128} * 2^{2048} * 2^{128}$ attempts. Table 5.2 illustrates the symmetrical key size and the time required to crack it. In the non-hierarchical method, suppose Eve wants to work out Alice's real ID from her non-hierarchical home index pseudonym (nHIP). Eve would not be able to recover the real ID from the nHIP because it is not generated based on Alice's real ID. In this case, Eve needs access to the healthcare provider server (HCP) to learn the relationship between the pseudonym and Alice's real ID,

which is a challenging task. The same is true if Eve wants to work out Alice's real ID from non-hierarchical home request pseudonyms, non-hierarchical foreign index pseudonyms, or non-hierarchical foreign request pseudonyms.

Theorem 4. *Methods are protected against repudiation attacks.*
Proof. The patient cannot deny the generation of a pseudonym as any pseudonym is generated using a secret key. The secret key is generated based on the private key of the patient. Thus, it is very hard for Alice to claim that Eve generated a message on her behalf.

Theorem 5. *Methods are protected against linkability attack.*
Proof. As we know, a patient has different index pseudonyms one for each data collection server. In addition, for each session with the same server, the patient uses a fresh request pseudonym. Each request pseudonym is generated based on the index pseudonym or a public key. Linking these pseudonyms correctly to the same patient is a challenging task as will be shown in the following section.

5.6.3 Degree of Anonymity

The degree of anonymity provided by HPGL and nHPGL methods is measured by using the Shanon entropy method [47]. What we need to show is that the patients are indistinguishable from the attacker. If the attacker could determine a particular patient to be the generator of a pseudonym by any means, the anonymity provided by the method is low.

To calculate the degree of anonymity provided by both methods, we assume that (X) is a discrete random variable which represents a pseudonym. This pseudonym can be correctly linked to a certain patient (i) from a set of (N) patients. In mathematics, this can be represented as ($p_i = Pr(X = i)$), where Pr is the probability. The current entropy (H(X)) of the corresponding pseudonym can be calculated as:

$$H(X) = -\sum_{i=0}^{n} P_i \log(p_i) \tag{5.1}$$

The maximum entropy of the system (H(M)) can be calculated as:

$$H(M) = \log N \tag{5.2}$$

The degree of anonymity (d) provided by both methods can be calculated as:

$$d = H(X)/H(M), 0 \leq d \leq 1. \tag{5.3}$$

1. When d = 0, the attacker knows the generator of the pseudonym (p_i) with probability 1.
2. When d = 1, all patients appear as being an originator with the same probability.

Suppose that 100 patients (i.e., N = 100) are using our methods. Each patient generates several pseudonyms and these pseudonyms can be hierarchical or non-hierarchical. The attacker cannot distinguish one particular patient as the owner of these pseudonyms. However, the attacker can divide the patients into say two groups (G_1 and G_2), G1 has 40 patients and G2 has 60 patients. Then the attacker assigns each group a probability of generating a set pseudonyms as follows.

$$G_1 = Pr/40, 0 \leq i \leq 40 \quad G_2 = 1 - Pr/60 \quad 41 \leq i \leq 60 \tag{5.4}$$

In Equation 5.4, we have two groups of patients, one with 40 patients and the other with 60 patients. Patients belonging to the same group are seen by the attacker as having the same probability of generating the pseudonyms.

Figure 5.6 shows that the degree of anonymity (d) is proportional to the number of patients. When the number of patients is 100, the maximum degree of anonymity (d = 1) is achieved for the probability distribution (p = .41). The degree of anonymity is equal to 0.8 when one group is assigned the probability p = .93 and the number of patients is (N=10,000). However, the anonymity does not drop to zero even in the case that we have only two patients in the system. This is because the attacker sees all patients as potential owners of the pseudonyms.

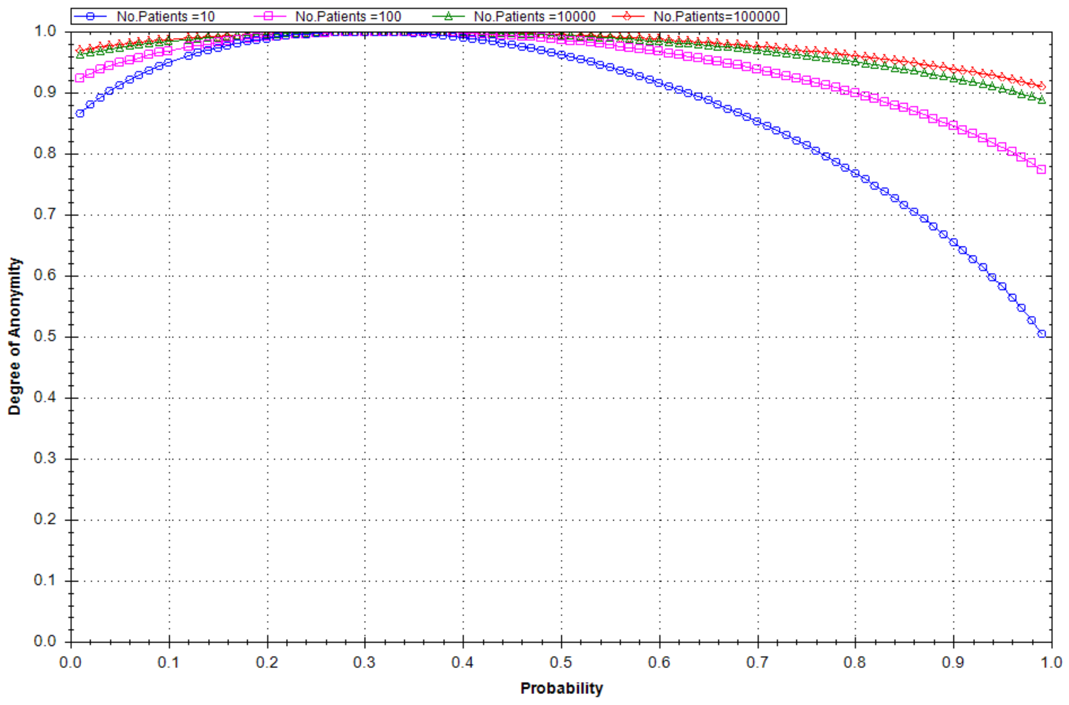


Figure 5.6: Degree of anonymity

5.6.4 Performance Analysis

This section presents a theoretical performance analysis of our hierarchical and non-hierarchical methods. The performance is measured by the computational cost required to generate and link pseudonyms. Table 5.3 summarises the computational cost for generating, verifying, and linking the hierarchical pseudonyms, and Table 5.4 summarises the computational cost

Table 5.3: Hierarchical crypto operations

Algorithm	Crypto Operation	Key size	Pseudonym size (Byte)	Time (ms)
HIPs-Gen	2 AES ENC	128	32	0
HIPs-Lnk	2 AES DEC	128		0
HRPs-Gen	1 AES ENC	256	48	0
HRPs-Lnk	1 AES DEC	256		21
FIPs-Gen	1 RSA ENC	2048	256	0.1
FIPs-Lnk	1 RSA DEC	2048		3.5
FRPs-Gen	1 AES ENC	256	272	0
FRPs-Lnk	1 AES DEC	256		21

Table 5.4: Non-Hierarchical crypto operations

Algorithm	Crypto Operation	Key size	Pseudonym size (Byte)	Time (ms)
nHIPs-Gen	EKeyGen	128	32	1.4
nHRPs-Gen	1 RSA ENC	2048	256	0.1 ms
nHRPs-Lnk	1 RSA DEC	2048		3.5 ms
nFIPs-Gen	1 RSA ENC	2048	256	0.1
nFIPs-Lnk	1 RSA DEC	2048		3.5
nFRPs-Gen	1 RSA ENC	2048	256	0.1 ms
nFRP-Lnk	1 RSA DEC	2048		3.5 ms

for generating, verifying, and linking the non-hierarchical pseudonyms.

The software used to implement the generation and linkage algorithms in both methods is Java 2 Platform, Standard Edition (J2SE). Java is chosen because it supports a set of standard security interfaces. We used a desktop computer running Windows 10 with a 1.99 GHz Intel Core i7 and 8GB of RAM. The timing results from the algorithms execution presented here are based on this computer's specifications.

5.6.4.1 Generation Computational Cost

Here we focus on the computational cost used for the generation of the hierarchical and non-hierarchical pseudonyms. Regarding the hierarchical method, the patient's mobile device is responsible for generating the foreign index pseudonyms, home and foreign request pseudonyms, while HCP is responsible for generating the home index pseudonym. In the non-hierarchical method, the patient's mobile device is responsible for generating all the pseudonyms.

Hierarchical Method

From Table 5.3, it can be seen that each patient has four types of of hierarchical pseudonym which are generated using cryptographic operations. For the generation of the home index pseudonym (HIP$_i$), two AES symmetrical operations (encryptions) are used. Using symmetrical operations as much as possible allows us to keep the computational cost as low as possible. The execution time taken to generate one home index pseudonym is negligible. To calculate the size of the pseudonym that results from the AES encryption, we use the equation 5.5. The AES has a fixed block size of 16 bytes regardless of key size. The home index pseudonym size is 32 bytes.

$$\text{PseudonymSize} = \text{InputSize} + (\text{BlockSize} - (\text{InputSize} \mod \text{BlockSize})) \tag{5.5}$$

For the generation of a foreign index pseudonym ($\text{FIP}^{\text{f}}_{i}$), one asymmetrical RSA encryption is needed. Since the number of foreign index pseudonyms is dependent on the number of foreign servers that the patient is going to use, the mobile device performs (n) RSA encryptions, where (n) is the number of foreign servers. The execution time taken to generate one foreign index pseudonym is 0.1 ms. The size of the foreign index pseudonym is 256 bytes. This is because we used RSA encryption. In RSA, the pseudonym size should always equal the size of the modulus (e.g. a 2048 key bit gives a 2048 bit output or 256 bytes).

For the generation of a foreign request pseudonym ($\text{FRP}^{\text{f}}_{i,r}$), one symmetrical AES operation (encryption) is needed. Since the number of foreign request pseudonyms is dependent on the number of the uploading requests, the mobile device performs (n) AES encryption, where (n) is the number of requests. The size of this pseudonym is 272 bytes. The execution time taken to generate one foreign request pseudonym is negligible. However, assuming that the patient needs to upload every second; then s/he needs 60 pseudonyms per minute. The execution time taken to generate them is 0.006 ms.

Non-Hierarchical Method

From Table 5.4, it can be seen that each patient has four types of non-hierarchical pseudonyms. For the generation of the home index pseudonym, the patient generates an elliptical curve public key using the EKeyGen algorithm. The size of this pseudonym is 32 bytes. The execution time taken to generate this pseudonym is 1.4 ms.

For the generation of a non-hierarchical foreign index pseudonym ($\text{nFIP}^{\text{f}}_{i}$), we need one RSA encryption and one ECC key generation. Since the number of non-hierarchical foreign index pseudonyms is dependent on the number of foreign servers that the patient is using, the mobile device performs (n) ECC key generation and RSA encryption, where (n) is the number of foreign servers. The execution time taken to generate one foreign index pseudonym is 1.5 ms. The size of the foreign index pseudonym is 256 bytes.

For the generation of a non-hierarchical foreign request pseudonym ($\text{nFRP}^{\text{f}}_{i,r}$), one asymmetrical RSA operation (encryption) is needed. Since the number of foreign request pseudonyms is dependent on the number of requests the patient needs, the mobile device may perform (n) RSA encryptions, where (n) is the number of requests. The size of this pseudonym is 256 bytes. The execution time taken to generate one foreign request pseudonym is 0.1 ms. However, if the patient needs to upload every second; then s/he needs 60 pseudonyms. The total execution time taken to generate them is 6 ms. The same is true for the generation of a non-hierarchical home request pseudonym (nHRP$_{i,r}$),

5.6.4.2 Linking Computational Costs

Here we focus on the computational cost for linking both methods. The HCP is responsible for linking the home index pseudonym to the patient's real ID. The home server is responsible for linking the foreign index pseudonyms to the patient's home index pseudonym. The home server is responsible for linking each home request pseudonym to the patient's home index pseudonym. Each foreign server is responsible for linking each patient's foreign request pseudonym to the patient's foreign index pseudonym. It should be noted that the linking algorithm requires two crypto operations, the decryption, and verification of the MAC associated with the message.

Hierarchical Method

From Table 5.3, it can be seen that the pseudonyms are linked using cryptographic operations. To link the home index pseudonym (HIP$_i$), the HCP uses two AES symmetrical operations (decryptions). The execution time taken to link one pseudonym is negligible. For linking a foreign index pseudonym (FIP$_i^{\mathrm{f}}$), two cryptographic operations are needed, asymmetrical RSA decryption (3.5 ms) and MAC verification. The execution time taken to link one foreign index pseudonym is 3.5 ms + HMAC$_{\mathrm{Ver}}$, where HMAC$_{\mathrm{Ver}}$ is a variable depending on the size of the message.

For linking a home request pseudonym (HRP$_{i,r}$) by the home server, one symmetrical AES operation (decryption), and a MAC verification are needed. Since the number of home request pseudonyms is dependent on the number of requests coming from patients, the home server may perform a number of AES decryptions and MAC verifications. The time taken to link one home request pseudonym is 21 ms. This is because to link each home request pseudonym to the correct home index pseudonym of the patient, the home server should first find the key that is used as an input to HRPs-Lnk algorithm. To do so, the home server needs to specify the segment that contains the correct home shared key of the patient. Then, the home server tries all the keys in the segment (in our experiment we specified that each segment contains 100 rows) to verify the MAC. If the server finds the key that verifies the MAC of the message, the same key is used to perform the decryption to find the home index pseudonym of the patient. The home server should do this process for a number of requests from different patients. This is also the process for the foreign request pseudonym.

Non-Hierarchical Method

As can be seen in Table 5.4, non-hierarchical pseudonyms are linked using decryption and MAC verification operations. To link the non-hierarchical home index pseudonym (nHIP$_i$), the HCP uses a mapping table. This table maps each non-hierarchical home index pseudonym to the patient's real ID. The execution time is 3.5 ms + HMAC$_{\text{Ver}}$, where HMAC$_{\text{Ver}}$ is a variable depending on the size of the message. According to [48] the data size per hour can reach up to 471 KB, which would require 2.1 ms to conduct the verification. It takes less time for a smaller size of message.

To link a non-hierarchical foreign index pseudonym (nFIP$_i^{\text{f}}$) by the home server, one asymmetrical RSA decryption and one MAC verification are needed. Since the number of non-hierarchical foreign index pseudonyms is dependent on the number of foreign servers that the patient is using, the home server of the patients performs (n) RSA decryptions and verifications, where (n) is the number of non-hierarchical foreign index pseudonyms. The execution time taken to link one foreign index pseudonym is 3.5 ms + HMAC$_{\text{Ver}}$, while the time to do HMAC$_{\text{Ver}}$ depends on the size of the message.

To link a non-hierarchical home request pseudonym (nHRP$_{i,r}$) by the home server, one asymmetrical RSA decryption and one MAC verification are needed. Since the number of non-hierarchical home request pseudonyms is dependent on the number of requests coming from patients, the home server may perform a number of RSA decryptions and MAC verifications. The execution time taken to link one home request pseudonym is 3.5 ms + HMAC$_{\text{Ver}}$, while the time required to perform HMAC$_{\text{Ver}}$ is dependent on the size of the message. The same is true for the foreign request pseudonym.

5.7 Chapter Summary

This chapter has first provided a survey of existing work in pseudonym generation. It has shown that none of the current works supports a pseudonym generation method that enables the patient to carry two different types of pseudonyms: one works as an account ID, and the other works as a session ID. It has presented two novel methods to support generating and linking different types of pseudonym; Both methods fulfil important requirements, which have not been considered in the relevant work. The hierarchical pseudonym generation and linkage method use a symmetrical cryptosystem to generate and link the pseudonyms. The method suffers from a high verification time. This motivated us to design the second method which is the non-hierarchical pseudonym generation and linkage. The second method uses an asymmetrical cryptosystem to generate and link pseudonyms and the time it takes for verification is much shorter than the first method.

Chapter 6

Anonymous Distributed Authentication (ADA)

6.1 Chapter Introduction

This chapter presents a design and evaluation of novel secure protocols for distributed authentication, known as Anonymous Distributed Authentication (ADA) protocols. There are two types of ADA protocols, one for registration across multiple domains and the other for session authentication in one domain (a data collection server). The evaluation of the ADA protocols is performed in two stages. First, informal security and privacy analysis of the protocols is performed to demonstrate the protocol's resilience against known attacks. Second, the evaluation of the protocols is performed using queuing theory.

This chapter is organized as follows. Section 6.2 reviews the literature on anonymous authentication. Section 6.3 gives a design preliminary of ADA protocols. Section 6.4 describes the details of the ADA protocols. Section 6.5 analyzes the protocols against the design requirements. Section 6.6 analyzes the protocols against the security and ID privacy requirements. Section 6.7 evaluates the performance of the proposed protocols. Section 6.8 describes the evaluation of the performance of the protocols using queuing theory. Section 6.9 compares the ADA to state-of-the-art anonymous distributed solutions. Finally, in Section 6.10, a summary of the chapter is provided.

6.2 Anonymous Authentication: A Literature Review

The architecture of our framework SPDC - as seen in chapter 4 - resembles the architecture of Federated Identity Management (FIM) systems [40]. However, the SPDC is different in that it has two identity providers: one is static and the other dynamic. The static one is the HCP and the dynamic is the home server. The patient can change the home server, say, every day, so one of the data collection servers will be selected by the patient to be the home server. The HCP provides the patient with a pseudonym certificate to register with the home server. The role of the home server is (1) blindly sign pseudonym certificates generated by the patient to access foreign servers, and (2) collect the patient's data that is scattered across foreign servers.

The FIM system has several models: the centralized identity model, the user-centric identity model, and the decentralized identity model. In this section, we will discuss each model.

6.2.1 The Centralized Identity Model

The centralized identity model [40] relies on a central entity to conduct the authentication between two entities. This model consists of users, service providers, and an identity provider. In this model, a user can request a service from a service provider anonymously as the real identity of the user is hidden from the service provider. However, before requesting a service, the service provider should authenticate the user. To conduct authentication in an anonymous way, the service provider redirects the user to his/her identity provider. The user performs the authentication with the identity provider using an authentication mechanism such as username/password or digital signature. After that, the identity provider issues an assertion to the user, which is a statement that contains authenticated information about the user. The user forwards the assertion to the service provider. The service provider authenticates the user and provides the requested service to the user. An example of this is accessing IEEE explorer, whereupon one is redirected to one's institute for authentication. OpenID [49], Shibboleth [50], and U-Prove [51] are examples of federated identity management systems. However, there are concerns about these systems related to the privacy of the user. They allow the identity provider to link user accounts across different service providers and track the user's activities. To solve this issue, the current federated systems suggest using a blind signature technique. This technique (i.e., the blind signature) prevents identity providers from tracking the user's activities.

One of the early federated identity management systems that applied this approach was PseudoID [52]. In this work, the identity provider issues the user a number of credentials that are signed blindly using the identity provider's private key. When the user wants a service from a service provider, the user's device sends one blind credential to the service provider, which then verifies the blind signature using the corresponding identity provider's public key. After the anonymous authentication, the service provider provides the requested service to the user. The blind signature in PseudoID is based on the RSA cryptosystem.

To prevent the identity provider from tracking the user, the author [53] presented a solution called a Crypt-book, as seen in Figure 6.1. The Crypt-book is a layer between a number of identity providers and service providers, which hides the real identity of a user from the service providers. The Crypt-book prevents identity providers from linking multiple accesses from the same user to different service providers. The Crypto-Book achieves these functions through two layers, the first of which involves credential producers and the second credential consumers (service provides). The credential producers produce multiple credentials for the user to be used with multiple service providers. The user shows one credential to each service provider. After that, the service provider creates a pseudonym account for the user. Specifically, the user first conducts the authentication with a number of credential producers using an authentication protocol (e.g. OAuth). After the authentication, the user chooses a value (r)

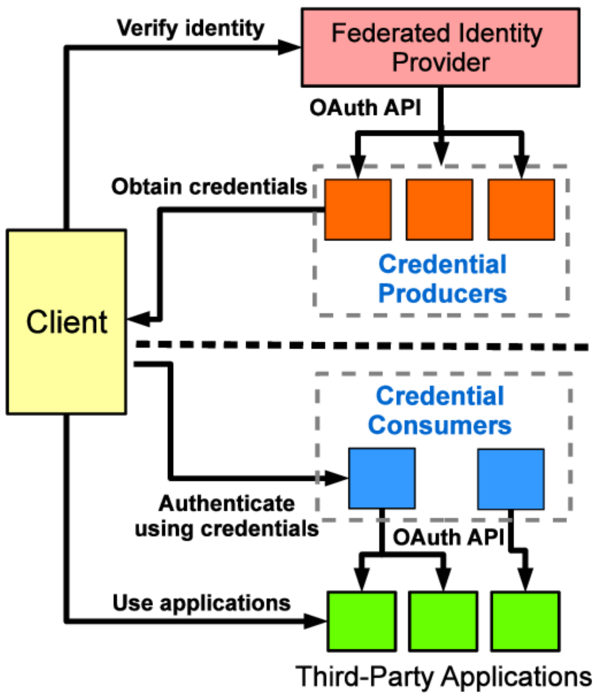


Figure 6.1: Crypt-book federated system [53]

to serve as a credential identifier. The user binds the r value to the service provider identity (ID_s) which the user wants to use, i.e. m =(r,ID_s). The user then requests a blind signature on the message (m) from a number of producers. Each producer generates a blind signature on the message and returns it to the user. After this, the user waits for a successful response from at least (n) producers. The user then unblinds the signatures to obtain a vector of signatures, which serves as the credential. The user sends the credential to the service provider which verifies the credential. If the verification result is positive, the service provider creates a pseudonymous account for the user. The drawback of the [53] is that the user needs to wait for a successful response from at least (n) producers.

Another solution suggested by [54] is called UnlimitedID. This is a method that allows a user to have multiple long term pseudonyms on different service providers without the identity provider being able to link these pseudonyms to the same user. The method suggests using anonymous attribute credentials based on algebraic message authentication codes (aMACs). The following explains how UnlimitedID works. First, the user generates a credential that has a secret key (SK) unknown to the identity provider, key-value attributes and an expiration time. The user first blinds the secret key and proves knowledge of the key by using non-interactive, zero-knowledge proof. The result of this step is credential encoding along with aMACs. Second, the user requests a service from a service provider. The service provider redirects the user to the identity provider for authentication. The user selects a number of attributes to be revealed to the service provider, blinds the attributes, and sends them along with aMACs to the identity provider. Using non-interactive zero-knowledge proof, the identity provider checks the authenticity of the user and creates a temporary pseudonym entry using the revealed attributes. Then, the identity provider redirects the user to the service provider along with the authentication code. The user passes the authentication code to the service provider. The service provider uses the authentication code to request an access token from the identity provider. The drawback of this solution is that the authentication is not directly between the user and the service provider. These solutions [53], [54] create a performance bottleneck by relying on one entity to generate the credentials for many users.

In [55], the authors proposed a user-centric identity management framework which allows a user to have a number of credentials for multiple purposes from multiple attribute authorities. If the user wants a service from a service provider (e.g. a bank), the service provider will ask the user to prove that s/he has a number of attributes (e.g. if a student has sufficient money). The user will ask a number of identity providers to issue verifiable credentials for each of the requested attributes. For example, the user may ask the university to prove that s/he is a student and may ask the bank to prove that s/he has sufficient money. Each identity provider issues credentials to the user, which are passed on by the user to the service provider.

A peer-to-peer system is proposed by Bianchi et al. [56], in which each service provider plays the role of both a service provider and a blind credential issuer. In [56], the user first reveals his/her real identity to one of the providers, say (SP_1), which issues the user a credential signed blindly. The user uses the credential to authenticate with another service provider, say (SP_2), from which the user can request another credential signed blindly by (SP_2) to access another service provider, say (SP_3). In all the credentials the user is responsible for generating the pseudonyms.

For a continuous health monitoring service, the patient needs to upload his/her data periodically (e.g. every 5 minutes) to a data collection server. Each upload should be authenticated by the server to allow the patient to upload and the server to store the data in the patient's account. Requesting authenticated pseudonyms or credentials from a certified authority for each upload is not efficient. The user should be involved in generating pseudonyms and credentials as suggested by the user-centric identity model which is described in the next section.

6.2.2 The User-Centric Identity Model

In this model, the user is involved in generating credentials to access different service providers.

In [57], the user performs authentication with different service providers based on a set of certified attributes issued by the user's identity provider. In this model, the user can generate a number of specific pseudonyms, one for each service provider. In contrast to other models that force the user to show all their attributes (e.g. age, place of work, gender) to the service provider, this model allows the user to reveal only certain attributes to be authenticated. This is achieved by using a number of cryptographic building blocks such as Verifiable Random Functions (VRFs) and Blank Digital Signature (BDS). The VRF allows the user to generate a pseudonym for each service provider. The BDS allows the user to generate a signature on a subset of attributes which can be verified by the service provider.

Another work proposed by [58] is called the SPICE framework. In this framework, the authors presented a novel authentication mechanism for a Digital Identity Management (DIM) system. A typical cloud-based DIM system consists of: Cloud Service Providers (CSPs), Identity Providers (IdPs), registrars, and users (cloud clients). The IdP function is to assign attributes to users. The registrars are used to verify the attributes given by the IdP, and then issue certificates to the users. Based on the certificates, the users can authenticate themselves to CSPs and gain access to authorized services. In SPICE, the registrar issues only one credential to each user no matter how many service providers the user wants to use. For authentication, the user generates (based on the credential) many certificates to prove the possession of (different sets of) attributes required by different service providers, without asking the registrar to issue a new certificate each time. This process randomizes the signature and hides attributes. By randomizing the signature, one signature can look different for multiple uses.

As one can see, by using this model the mobile device has to store credentials to access service providers. To overcome this problem, the next model enables decentralized nodes to store credentials using blockchain technology. This will help the user to conduct authentication in any domain, even if the domain is certified by a different certificate authority.

6.2.3 The Decentralized Identity Model

In this model, the power is given to a collection of nodes that use distributed ledger technology (DLT) to store identity information about users where no node can change the content of DLT. The trust between nodes is reached by a consensus mechanism. The distributed DLT is built using blockchain technology.

A blockchain is a set of blocks that are used to store verified requests or transactions. These blocks are connected in chronicle order. This results in a fully traceable history of events. Each block contains a header and a body. The header contains the ID, nonce, and a hash of the previous block, while the body contains information about the request or the transaction. To generate a block for each request sent by a user and to connect it to the blockchain, there are special nodes called miners. The miner creates the block for each request through an algorithm called a consensus algorithm. Each miner has to ensure that the request sent by the user is verified. This is done by looking the request up in the blockchain to see whether the user has accomplished the request or not. After the verification, each miner generates the block for the request and posts it to the blockchain. To ensure that only one block is posted to the blockchain network, each miner is required to solve a puzzle through algorithms known as consensus algorithms. One of these algorithms is called the proof of work (PoW) algorithm, which can be explained as follows. Each miner should come up with a hash that provides enough leading zeroes. By seeing enough leading zeroes, we can be sure that it took a large amount of computational effort (which is why it is known as PoW) and only one block is posted.

A plethora of research has been performed to solve the problem of cross domain authentication using blockchain technology explained above. One of these works is Trustroam [59] which proposed an authentication method to provide cross-domain roaming authentication.

This means that a user from institute A can use his/her identity credentials to access a network of institute B. In contrast to existing solutions such as Eduroam, the authentication of this proposed solution is based on distributed consensus algorithms of the blockchain. In this solution, multiple institutes can form a blockchain network to authenticate requests sent by multiple users. The detailed explanation of the proposed solution is as follows. First, in the blockchain network there is an address for each entity in the system. This address works as an identity for each entity. This address is an elliptical curve public key. This address is used to identify each entity across multiple institutes. Now, when a user of institute A requests access to a network of institute B, the user sends the request signed by the corresponding private key. The request is received by institute B which verifies the digital signature and issues an authentication code to the user. Then, institute B broadcasts the authentication code (AuthCode) to all blockchain nodes. Each node that receives the AuthCode will be provoked to authenticate the user. Each node will compete for the right to write the authentication on the blockchain under the PoW consensus algorithm. This method allows institutes to link different accesses by the same user as the user is using the same identity across domains.

To solve the linkability problem, the authors proposed blockchain lightweight anonymous authentication (BLA)[60]. The proposed mechanism allows a vehicle to access services across distributed domains. This mechanism allows the vehicle to use different pseudonyms each time to prevent linking multiple requests sent by the same vehicle. Each domain contains a Service Manager (SM), Witness Peer (WP), and Road Side Unit (RSU). When the vehicle wants to use a service provided by a domain, it sends the request message to the RSU in that domain. To make the request unlinkable across domains, the vehicle uses a fresh pseudonym for each request to the RSU. The RSU in turn sends the request to the SM of the domain, which first proves the authenticity of the request by deriving the real identity of the vehicle from the message. It then authenticates the vehicle and sends the result to the all WPs which use a consensus algorithm to write the authenticating result to the blockchain. Then, the RSU is informed to provide the service for the vehicle. Next, when the vehicle moves to another region of a new RSU, the vehicle can select no re-authentication and the vehicle can be authenticated using the same pseudonym.

Other authors proposed a solution called a BlockCAM [61], to solve a traditional problem in cross-domain authentication. This problem occurs when a user U in domain A wants to access a service in domain B the root certificate authority (CA) in both domains (i.e., A and B). The user should have a mutual authentication or both domains should be certified by a third-party certificate authority. Traditional authentication suffers from a high cost of certificate management and a network bottleneck. In contrast to the traditional approach, the BlockCAM leverages the blockchain network in which each CA root works as a verification node. In this blockchain network, a collection of CAs attempts to verify a certificate submitted by a user. Once the certificate is verified, the CA generates a block using the consensus algorithm. Each block stores the hash of the user's certificate and the status of the certificate. The CA then posts the block to the network. When a user U wants to access a service from a domain B, U sends his/her certificate to B. B then looks up the blockchain records to find

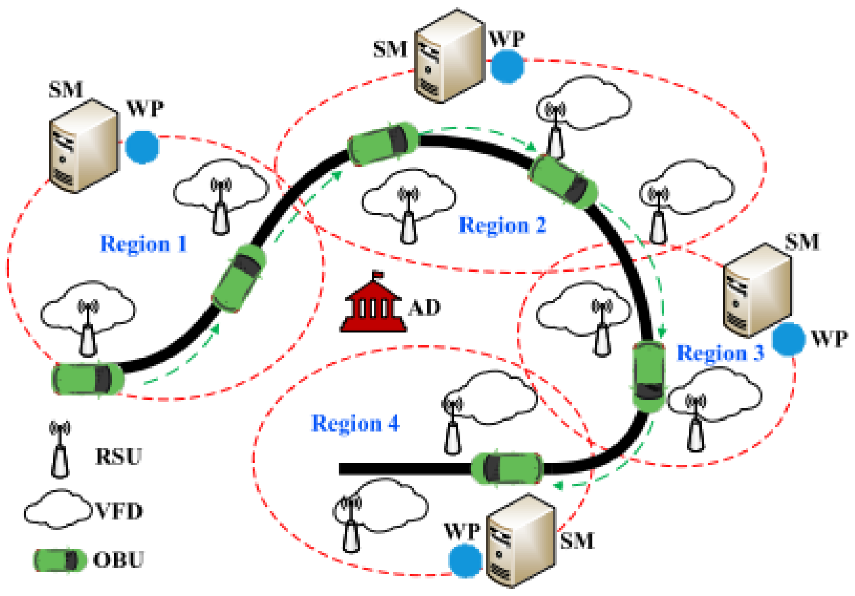


Figure 6.2: The BLA system [60]

U's certificate. Subsequently B generates the hash of the certificate and compares it with the hash stored in the block. If both hashes are the same, B verifies U and provides the requested service.

Another work proposed by [62] suggested unlinkable authentication. The architecture of this proposed solution consists of multiple domains each of which has a service manager (SM), roadside infrastructure (RSU) which facilitates communicating with the SM, and vehicles that register with one SM and communicate wirelessly with RSUs. All the SMs from different domains form a blockchain network. The vehicle first selects one SM as a home server to register using its real ID. The SM_h generates a blockchain ID for the vehicle based on its real ID (BID_v). The SM sends the BID_v along with other information to the blockchain. When the vehicle moves across domains, it receives a broadcast message from a number of RSUs, the vehicle can determine the public key of each SM based on the received broadcast. If the vehicle decides to authenticate with SM_i, the vehicle first uses the public key of SM_i to encrypt its blockchain ID (BID_v). Then, the vehicle uses another parameter to generate a pseudonym (PS_v) for each request. The vehicle sends the encrypted BID_v and the generated PS_v to the SM_i. The SM_i decrypts the encrypted BID_v to obtain the vehicle's blockchain ID (BID_v). Then uses the BID_v to obtain information about the vehicle from the blockchain. It uses the obtained information to decide whether the generated pseudonym (PS_v) is correct or not. This proves the credibility of the vehicle.

As explained in previous chapters, we aim to support anonymous yet linkable data uploads by the patient on any data collection server. To do so, we need two mechanisms. One is anonymous registration across different domains (different data collection servers), and the other is anonymous authentication inside the domain (i.e., a data collection server). This aims to (i) enable each data collection server that worked as a home server to anonymously link their patients' data which is scattered across different foreign servers, (ii) allow each data collection server which worked as a foreign server to link multiple uploads to the same patient's account.

In this chapter, we detail the design of ADA, which consists of an anonymous inter-domain registration and intra-domain authentication.

6.3 ADA Design Preliminaries

This section details the notations, assumptions, requirements, and high-level ideas used in the design of the ADA solution.

6.3.1 Design Requirements

The ADA is designed to satisfy the following requirements.

R1. Undetectable: This property means concealing the patient actions from any internal or external entity. This means that the healthcare provider (HCP) does not know which data collection servers the patient will use as home and foreign servers.

R2. Inter-domain registration unlinkability: This property means concealing correlations between multiple actions performed by the same patient across distributed data collection servers. This means that no entity can establish a connection between a patient and his/her index pseudonyms and certificates which are used in the registration with other data collection servers.

R3. One time registration certificate: The certificate which is used to register with a data collection server should only be used once. This means that in the future and for the re-registration with the same server, the patient should use a new certificate.

R4. Intra-domain authentication unlinkability: This property means concealing correlations between repeated actions performed by the same patient with the same data collection server. For example, only the foreign server to which the patient is currently uploading should be able to link multiple uploads to the same patient.

R5. Stateless request authentication: The authentication per request should be done without the need for the server to search for the shared key in its database to verify the request.

R6. Patient-centric: The patient should be involved in the generation of the credentials (i.e., certificates, pseudonym IDs).

6.3.2 Threat Models

In this section, we present and discuss the threat models that ADA should be protected against.

- Impersonation: Is where an unauthorised entity pretends to be authorised (e.g. a patient, a data collection server) to gain access to a resource and patient privacy information. Impersonation usually occurs when an attacker successfully steals the credential of an authorised entity.

- Brute force attack: Is a strategy that involves checking all possible binary combinations of a secret (e.g., a key) until the correct one is found.

- Certificate forgery/theft: Occurs when an unauthorised entity generates a valid patient pseudonym, intending to deceive a data collection server and misuse the pseudonym to access sensitive patient data. The unauthorised entity could also obtain a valid pseudonym by stealing it from a legitimate entity, such as a trusted party or a patient.

- Replay attack: Is where an unauthorised entity records a transaction message and reuses it later to conduct a malicious action.

- Repudiation attack: Is where an entity takes part in a transaction then denies involvement in the transaction.

6.3.3 High-Level Ideas

In this section, we discuss the ideas used to design the ADA.

- **Idea 1.** The ADA is designed to use two types of credentials. The first is a digital certificate, which is used for registration across distributed data collection servers (inter-domain). There are two types of digital certificates: Home Pseudonym Certificates (HCerts) and Foreign Pseudonym Certificates (FCerts). The second type of credential is a fresh request pseudonym, which is used to authenticate the patient repeatedly (i.e., for each request) with the same data collection server.

- **Idea 2.** To satisfy the first requirement (i.e., being undetectable), the ADA is built as a certificate-based system. This means that before a patient requests a service from any data collection server, they should register with the server using a certificate designed for that particular server. The certificates are generated by different entities. Specifically, the patient requests the Home Pseudonym Certificate (HCert) from the HCP. This certificate enables the patient to register with the selected home server. Then, the patient generates the Foreign Pseudonym Certificates (FCerts) and request a blind signature on these certificates from the home server. These certificates enable the patient to register with foreign servers. The patient registers with the home and foreign servers directly using the certificates (there is no redirect authentication) and both types of certificate do not state any private information about the patient's real ID.

- **Idea 3.** To satisfy the second requirement of inter-domain unlinkability, the ADA is designed to use one certificate for each data collection server. The HCP issues the HCert to the patient to be used with the home server and when the patient changes home server, they no longer use the old HCert, but request a new HCert from the HCP to register with a new home server. In the same manner, when the patient needs to upload to any foreign server, the patient incorporates with the home server to generate foreign pseudonym certificates (FCerts), one for each foreign server. Each FCert is used to register with one foreign server. In addition, when the patient finishes uploading to the foreign server and after a time wants to register again with the same foreign server, the patient uses a new FCert.

- **Idea 4.** To satisfy the third requirement, which is the one-time show credential, the ADA restricts the patient to a single use of a certificate and must be within a time limit, in this case 24 hours.

- **Idea 5.** To satisfy the fourth requirement of intra-domain unlinkability (repeated authentication with the same data collection server), we use the MAC verification method. The patient uses a fresh request pseudonym, one for each request with the same data collection server. The data collection server can link different pseudonym IDs for the same patient by verifying the MAC associated with the pseudonym.

- **Idea 6.** To satisfy the fifth requirement, which is stateless session authentication, the ADA uses non-hierarchical pseudonyms. By decrypting the non-hierarchical pseudonym, the server obtains the verification key that is used to verify the patient.

- **Idea 7.** To satisfy the sixth requirement, the ADA allows the patient to generate two types of credentials: foreign pseudonym certificates and request pseudonym IDs.

6.3.4 Assumptions

- The patient has registered with the HCP using his/her true identity and has obtained the Home Pseudonym Certificate (HCert).

- The communication between the patient and any data collection server takes place over the Internet using an anonymous communication channel.

- The patient has already selected the foreign servers and the home server. The strategy of the selection is out of the scope of the thesis.

- The index nonce is sent by the home and foreign servers to the patient to prevent replay attacks.

6.3.5 Credential Types

In this section, we introduce two types of credential, namely pseudonym certificates and request pseudonym IDs. The pseudonym certificates are Home Pseudonym Certificates (HCert) and Foreign Pseudonym Certificates (FCerts), both of which are elliptical curve certificates which use a short key length of 256 bits and the Elliptic Curve Digital Signature Algorithm (ECDSA).

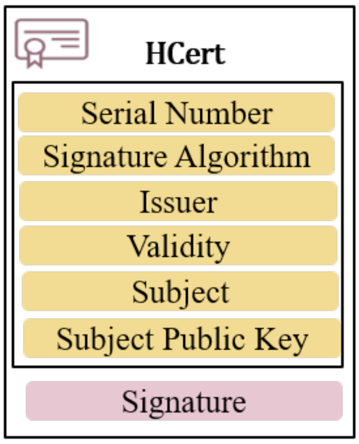


Figure 6.3: The home pseudonym certificate

6.3.5.1 Home Pseudonym Certificate

The home pseudonym certificate (HCert) is generated by the HCP. This certificate is used to register with the selected home server. The HCert as shown in Figure 6.3 contains the following information. The serial number, signature algorithm, issuer, validity, subject, the subject public key and the signature. The previous data structure fields are defined in Chapter 4. The following presents the values of some fields. The signature algorithm used is the ECDSA, the issuer is the HCP, the subject is the home index pseudonym (HIP), the subject public key is the elliptic curve public key of the patient, and the signature is the HCP's signature using its elliptical curve private key. This certificate is also used to authenticate the non-hierarchical home index pseudonym (nHIP).

6.3.5.2 Foreign Pseudonym Certificates

The foreign pseudonym certificate (FCert) is generated by the patient and blindly signed by the patient's home server. The FCert is used to register the patient to a foreign server. Each foreign server has an FCert. The FCert as shown in Figure 6.4 contains the signature algorithm, the home ID, validity, subject, the subject public key and the signature. The signature algorithm is the blind signature, the home ID is the ID of the home server of the patient, the subject is the foreign index pseudonym (FIP), the subject public key is the temporal elliptic curve public key of the patient, and the signature is the blind signature of the home server using its elliptical curve proxy private key. This certificate is also used to authenticate the non-hierarchical foreign index pseudonym (nFIP).

6.3.5.3 Request Pseudonym IDs

The request pseudonyms are generated by the patient for two purposes: to prevent linking multiple transactions performed by the patient by an unauthorized entity and to be used as a credential to verify the authenticity of the patient. The request pseudonym can be hierarchical and non-hierarchical. To verify the hierarchical request pseudonym, the server needs to search for the verification key in its database. To verify the non-hierarchical request pseudonym, the

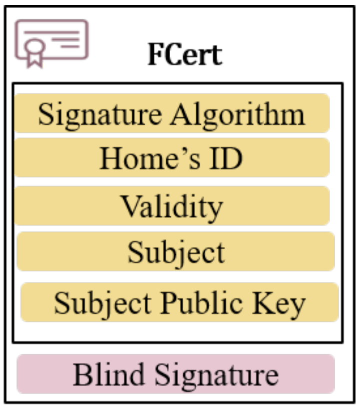


Figure 6.4: The foreign pseudonym certificate

server does not need to search in the database for the verification key; instead it decrypts the non-hierarchical request pseudonym to find the elliptical curve public key. Then, the server multiplies the elliptical curve public key with the server's elliptical curve private key. The multiplication process results in the shared key.

6.4 ADA Protocols

Before the description of the protocols, we introduce the system initialization and the message formats of the protocols.

6.4.1 System Initialization

The healthcare provider server (HCP) initializes the system by establishing the domain parameters which define the asymmetrical, symmetrical, and elliptic curve cryptosystems. All the entities of the system download these parameters from the HCP. Each patient, along with each data collection server (DCsrv), generates an ECC public/private key pair, EPK_i/EPR_i and EPK_{dc}/EPR_{dc}, respectively. The public keys are signed by the HCP and certified in the form of digital certificates. In addition, each data collection server generates RSA public/private key pairs, RPK_{dc}/RPR_{dc} of different sizes (i.e., 15360 and 2048). These public keys are signed by the HCP and certified in digital certificates. A list of valid data collection server certificates is stored in an open-access database. The patient's mobile device can access and download a number of certificates for the data collection servers with which it wants to establish communication. It should be noted that, even if the DCsrv is an intruder, it can not learn anything about the patient's data as the data is encrypted.

6.4.2 Protocol Message Formats

There are two protocols, one for inter-domain anonymous registration and the other for intra-domain anonymous authentication. Regarding inter-domain anonymous registration protocols, there are two types of messages, a request message (Req-Msg) and a response message

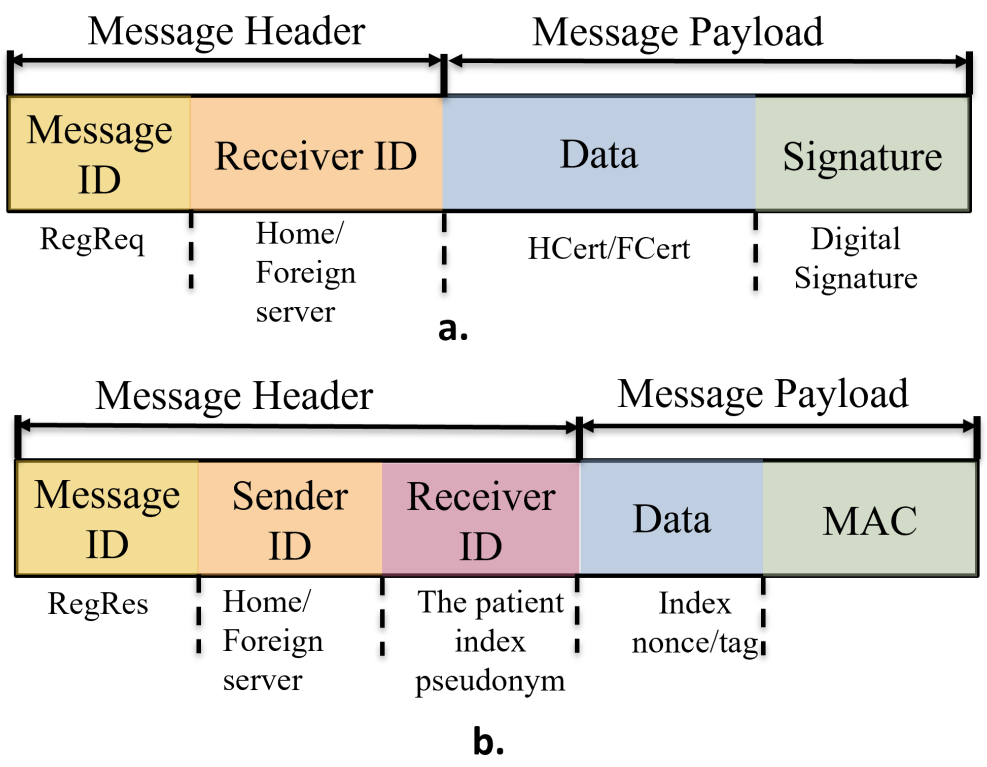


Figure 6.5: Registration protocol format (a) request (b) response

(Res-Msg). The message format for the request and response message is shown in Figure 6.5. The message format consists of a header and a body. The header contains the message ID and the receiver ID, while the body contains the payload and the digital signature. The message ID is a registration request, the receiver ID is the ID of the home server or the foreign server. The payload contains the home pseudonym certificate (HCert) or the foreign pseudonym certificate (FCert), and the digital signature using the patient's private key. The header of the message contains the message ID, the sender ID, the receiver ID, while the body contains the payload and the MAC. The message ID is a registration response, the sender ID is the ID of the home server or the foreign server. The receiver ID is the index pseudonym of the patient, the payload contains the tag range and index nonce, and the MAC is generated using the home shared key or the foreign shared key. The message format for the request and response message is shown in Figure 6.6. With the intra-domain anonymous authentication protocol, the header of the request message contains the message ID, the sender ID, the receiver ID, the tag number (only in the case the request pseudonym is a hierarchical one), while the body contains the payload and the MAC. If the message ID is an upload request (UpReq), then the receiver ID is the foreign server ID, the sender ID is the foreign request pseudonym, the payload is the encrypted health data of the patient (2EMPGHD), and the MAC is generated using the patient's foreign shared key. If the message ID is a blind signature request (BlndSigReq), then the receiver ID is the home server ID, the sender ID is the home request pseudonym, the payload is the blind hash, and the MAC is generated using the patient's home shared key. The header of the response message contains the message ID, the sender ID, and the receiver ID, while the body contains the payload and the MAC. If the message ID is an upload response (UpRes), the sender ID is the ID of the foreign server, the receiver ID is a foreign

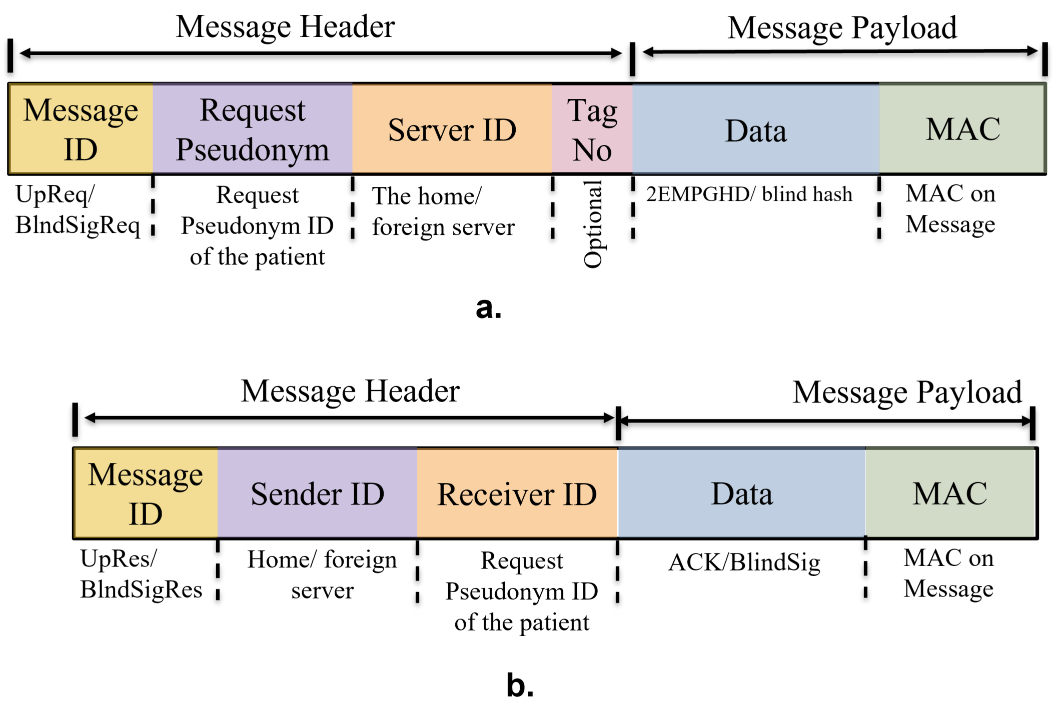


Figure 6.6: Other message protocol format (a) request (b) response

request pseudonym of the patient, the payload contains an acknowledgment, and the MAC is generated using the foreign shared key. If the message ID is a blind signature response (BlndSigRes), the sender ID is the ID of the home server, the receiver ID is a home request pseudonym of the patient, the payload contains a blind signature, and the MAC is generated using the home shared key.

6.4.3 Inter-Domains Anonymous Registration

Registration with home and foreign servers is done offline. The patient first performs registration with the home server using a protocol called the Home Server Registration (HSR) protocol. After the registration with the home server, the patient constructs a number of foreign pseudonym certificate (FCerts) using a protocol called the FCert Construction (FCertC) protocol. Then, the patient uses each FCert to perform the registration with each foreign server using a protocol called the Foreign Server Registration (FSR) protocol.

6.4.3.1 Home Server Registration (HSR) Protocol

This protocol registers and authenticates the patient with the selected home server. To begin the registration, the patient's mobile device constructs a home registration request (hRegReq) message and sends it to the home server. The home server receives the request message and responds to the patient with a home registration response (hRegRes) message.

To construct the hRegReq message the patient's mobile device does the following. The protocol is shown in Figure 6.7.

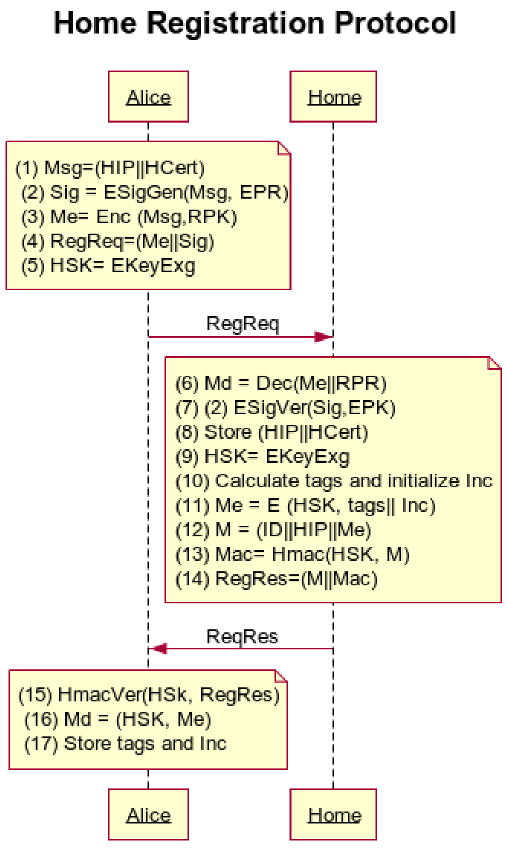


Figure 6.7: Home registration protocol

(1) The mobile device generates a message (M) which contains the patient's home index pseudonym (HIP_i) and home pseudonym certificate ($HCert_i$), i.e., $M = (HIP_i||HCert_i)$.

(2) It generates a digital signature (σ) on M, using the patient's elliptical curve private key (EPR_i). This private key corresponds to the elliptical curve public key (EPK_i) identified in the HCert. This digital signature is generated by using the ESigGen algorithm, i.e. $\sigma = \mathrm{ESigGen}(M, EPR_i)$.

(3) It encrypts the message (M) using the RSA public key of the home server (RPK_h). This step is done using RSA encryption algorithm, i.e. $Me = \mathrm{Enc}(M, RPK_h)$.

(4) It concatenates the digital signature with the encrypted message along with the ID of the home server (ID_h), and it sends the request message to the home server, i.e. $hRegReq = (ID_h||Me||\sigma)$.

(5) After the mobile device sends the request to the home server, the mobile device generates the Home Shared Key (HSK_i), using EKeyExg algorithm, i.e., $HSK_i = EPR_i * EPK_h$.

After the home server receives the hRegReq message, it does the following.

(6) The home server decrypts the encrypted part of the hRegReq message, using its RSA private Key (RPR_h). This is to extract the patient's home pseudonym certificate ($HCert_i$). This step is done using the RSA decryption algorithm, i.e., $Md = \mathrm{Dec}(Me, RPR_h)$.

(7) The home server performs three verifications. The first verifies the HCP signature (in the certificate) using HCP's elliptical curve public key (EPK_{HCP}). This step is done by using the ESigVer algorithm. The second verification is of the patient's digital signature (σ) using his/her EPK_i identified in the home pseudonym certificate by using the ESigVer algorithm. The third verification verifies that the home pseudonym certificate is within its validity period (our policy restricts the life of the certificate to 24 hours).

(8) After the verifications, the home server initializes the index nonce for the patient and stores both the home index pseudonym (HIP_i) and the certificate ($HCert_i$) in its database.

(9) The home server generates the home shared key (HSK_i) using the EKeyExg algorithm, i.e. $HSK_i = EPK_i * EPR_h$. This key is stored in the database. Then, it calculates the tag ranges (TagRng) using the CSB algorithm based on which segment in the database the patient's information is stored.

(10) After that, the home server prepares the home registration response (hRegRes) message (M) which contains the index nonce (Inc) and tag ranges (TagRng), i.e., $M = (Inc||TagRng)$.

(11) This message is then encrypted using the HSK_i, i.e. $Me = E(M, HSK_i)$.

(12) The home server concatenates its ID, and the patient's ID (HIP_i) to the encrypted message and generates a Message Authentication Code (MAC) using HSK_i, i.e., $MAC = HMAC(Me, HSK_i)$.

(13) The home server constructs the response message as follows: $hRegRes = (ID_h||HIP_i||Me||MAC)$. and sends it to the mobile device.

The mobile device receives the hRegRes message and performs the following.

(14) It verifies the MAC using the home shared key (HSK_i). This step is done using the HMAC algorithm. If the verification is successful, then the mobile device uses the same key to decrypt the encrypted part of the message, i.e., $Md = D(Me, HSK_i)$. Otherwise it discards the message.

(15) Finally, the mobile device stores the tag range. The tag range is used when the mobile device requests a blind signature from the home server. The tag range is used with the hierarchical request pseudonym.

6.4.3.2 Foreign Server Registration (FSR) Protocol

This protocol registers and authenticates the patient with/to a foreign server. The protocol has two messages, the foreign registration request (fRegReq) message and the foreign registration response (fRegRes) message. To begin the registration, the patient constructs the fRegReq message and sends it to the foreign server. The foreign server responds with a fRegRes message. Details of how these messages are constructed are provided below.

(1) The mobile device generates a message (M) which contains the foreign index pseudonym of the patient ($\mathrm{FIP_i^f}$) and the FCert certificate ($\mathrm{FCert_i^f}$), i.e., $\mathrm{M} = (\mathrm{FIP_i^f}||\mathrm{FCert_i^f})$.

(2) It then generates a digital signature (σ) on the message (M) using the ESigGen algorithm with temporal private key ($\mathrm{tPR_i}$). This private key corresponds to the temporal public key ($\mathrm{tPK_i}$) stated in the FCert certificate, i.e., $\sigma = \mathrm{ESigGen}(\mathrm{M}, \mathrm{tPK_i})$.

(3) The message (M) is encrypted using the RSA public key of foreign server ($\mathrm{RPK_f}$), i.e., $\mathrm{Me} = \mathrm{Enc}(\mathrm{RPK_f}, \mathrm{M})$.

(4) The mobile device constructs a registration request (fRegReq) message. This message contains the ID of the foreign server ($\mathrm{ID_f}$) and the encrypted message Me, i.e. $\mathrm{fRegReq} = (\mathrm{ID_f}||\mathrm{Me}||\sigma)$.

(5) The mobile device sends the request message to the foreign server.

(6) The mobile device generates the foreign shared key ($\mathrm{FSK_i^f}$) using the EKeyExg algorithm (i.e. $\mathrm{FSK_i} = \mathrm{tPR_i^f} * \mathrm{EPK_f}$).

The foreign server receives the fRegReq message and does the following.

(7) It decrypts the message using its private key ($\mathrm{RPR_f}$), i.e., $\mathrm{M} = \mathrm{Dec}(\mathrm{Me}, \mathrm{RPR_f})$.

(8) It extracts the patient's $\mathrm{FCert_i^f}$ and verifies the following: a) the home server's signature on the FCert using its proxy public key ($\mathrm{PPK_h}$) as an input to the blind signature verification (BlndSigVer) algorithm, b) the FCert is within its validity period (24 hours), c) the patient's digital signature using his/her $\mathrm{tPK_i^f}$ stored in the FCert.

(9) If verification is successful, the foreign server stores both the foreign index pseudonym ($\mathrm{FIP_i^f}$) and the patient's certificate ($\mathrm{FCert_i^f}$) in its database. Then, it initialises the index nonce for the patient.

(10) The foreign server generates a foreign shared key by using EKeyExg algorithm. (i.e., $\mathrm{FSK_i^f} = \mathrm{tPK_i^f}{*}\mathrm{EPR_f}$). Then, it stores the shared key in the database. It calculates the tag ranges (TagRng) using the CSB algorithm based on which segment in the database the patient's shared key is stored.

(11) The foreign server generates a message (M) which contains the index nonce (Inc) and the Tag ranges (TagRng), i.e., $\mathrm{M} = (\mathrm{Inc}||\mathrm{TagRng})$.

(12) The message (M) is then encrypted with the foreign key ($\mathrm{FSK_i^f}$), i.e., $\mathrm{Me} = \mathrm{E}(\mathrm{FSK_i^f}, \mathrm{M})$.

(13) The foreign server generates the MAC on the message (Me) using the foreign shared key ($\mathrm{FSK_i^f}$), i.e., $\mathrm{MAC} = \mathrm{HMAC}(\mathrm{Me}, \mathrm{HSK_i})$.

(14) The foreign server constructs the registration response message (fRegRes). The response message contains the ID of the foreign server ($\mathrm{ID_f}$), Me, the foreign index pseudonym of the patient ($\mathrm{FIP_i^f}$), and MAC, i.e., $\mathrm{fRegRes} = (\mathrm{ID_f}||\mathrm{FIP_i^f}||\mathrm{Me}||\mathrm{MAC})$. It then sends the response message to the mobile device.

The mobile device receives the fRegRes message and does the following.

(15) It verifies the MAC using the foreign shared key ($\mathrm{FSK_i^f}$).

(16) If the MAC is successfully verified, the mobile device uses the same key to decrypt the encrypted part of the message, i.e., $\mathrm{Md} = \mathrm{D}(\mathrm{Me}, \mathrm{FSK_i^f})$. Then, it stores the tag range and index nonce (Inc) to be used later when requesting an uploading service.

6.4.4 Intra-Domain Anonymous Authentication

Authenticating each request with home or foreign servers is done online. The authentication is done using the hierarchical or non-hierarchical request pseudonym. In this Section we will cover the authentication of each request with the home server for the purpose of constructing Foreign Pseudonym Certificates (FCerts). The authentication of each request with the foreign server for the purpose of uploading the data will be introduced in the next chapter.

6.4.4.1 FCert Construction (FCertC) Protocol using Hierarchical Pseudonym

This protocol allows a patient to request a blind signature from the home server to construct FCert. In this protocol one can see how the authentication can be done using the home requesting pseudonym without attaching the home pseudonym certificate (HCert) for each request. The steps of the authentication are shown in Figure 6.8.

(1) The patient's mobile device performs the following:

- Generates a temporal elliptic curve key pair ($\mathrm{tPK_i^f}$, $\mathrm{tPR_i^f}$) using the EKeyGen algorithm.
- Generates a foreign index pseudonym (FIP) using the FIPs-Gen algorithm.
- Generates the hierarchical home request pseudonym ($\mathrm{HRP_{i,r}}$) using the HRP-Gen algorithm.
- Generates the other data structure fields of the FCert which are: the signature algorithm, the home ID, and the validity. The mobile device uses the BlndMsg algorithm to generate blind factors. After that, it generates hash of the FCert data structure fields (e). Then, it blinds the hash result. It should be noted that the FCert data structure fields are the signature algorithm, the home ID, the validity, the subject (the foreign index pseudonym (FIP)), and the subject public key (temporal elliptical curve public key (tPK)).

(2) Then, the mobile device constructs the blind signature request message (BlndSigReq) which contains the hierarchical home request pseudonym ($HRP_{i,r}$), the ID of the home server (ID_h), the blinded hash (e), a tag number (TagNo), and a MAC generated on the previous fields using the home shared key (HSK_i). Subsequently, it sends the message to the home server, i.e., BlndSigReq=($HRP_{i,r}$||ID_h || TagNo || e || MAC).

The home server receives the request and does the following.

(3) It uses the TagNo as an input to the FVK algorithm. This is to search for the home shared key (HSK_i), which verifies the MAC.

(4) If the MAC is successfully verified, the home server uses the key as an input to the HRPs-Lnk algorithm. This is to find the home index pseudonym (HIP_i) and the index nonce.

(5) After finding the patient's HIP_i, the home server validates the index nonce. This validation guarantees the freshness of the home request pseudonym (to protect against replay attacks). It then checks that the HCert of the patient ($HCert_i$) has not expired. If both verifications are correct, the home server moves to the next step.

(6) The home server signs the blind hash (e) using the BlndSigGen algorithm.

(7) The home server constructs the blind signature response message (BlndSigRes), which contains the patient's request pseudonym ID ($HRP_{i,r}$), the home server's ID (ID_h), and blind signature (S^*). Then it generates the MAC on the message using (HSK_i) and sends the response to the patient's mobile device, i.e., BlndSigRes=(ID_h || $HRP_{i,u}$|| S^* || MAC).

The mobile device receives the BlndSigRes from the home server and does the following.

(8) It verifies the MAC associated with the message using his/her (HSK_i).

(9) It uses the Blind Signature Driven (BlndSigDrv) algorithm. This is to deduce the unblind signature from the blind signature (S^*).

(10) Then the mobile device attaches the unblind signature in the signature field of the FCert. The certificate is now ready to be used. The patient constructs as many FCerts as there are foreign servers.

6.4.4.2 FCert Construction (FCertC) Protocol using Non-Hierarchical Pseudonyms

The steps of the authentication are shown in Figure 6.9.

(1) The patient's mobile device performs the following.

- It generates a temporal elliptic curve key pair public and private key respectively (tPK_i^f, tPR_i^f) using the EKeyGen algorithm.

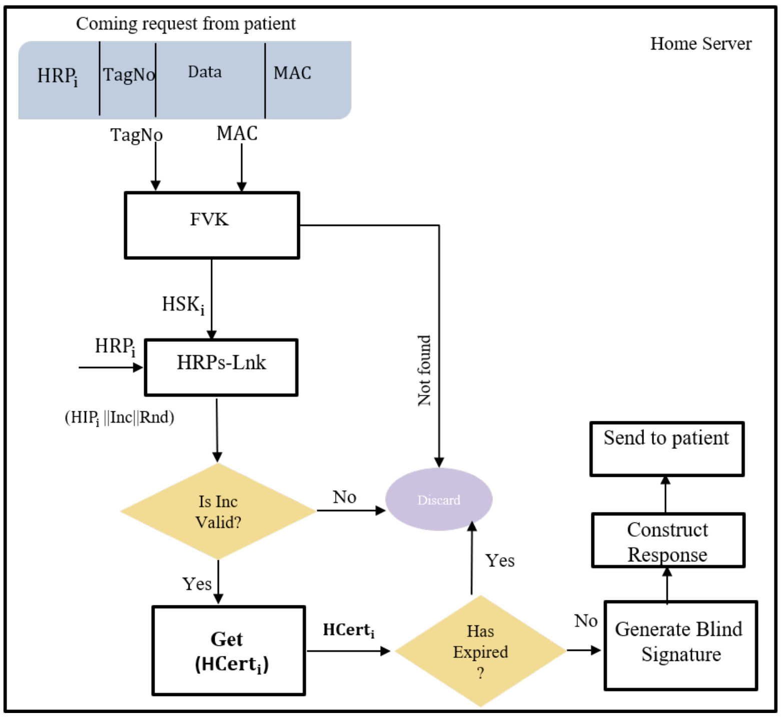


Figure 6.8: Authentication using hierarchical pseudonym

- It then generates the non-hierarchical home request pseudonym (nHRP$_{i,r}$) using the nHRP-Gen algorithm.
- It generates the other data structure fields of the FCert which are the signature algorithm, the home ID, and validity. Then it uses the BlndMsg algorithm to generate blind factors, hash the FCert data structure fields and then blind the hash result. The FCert data structure fields are the signature algorithm, the home ID, the validity, and the subject public key (temporal elliptical curve public key (tPK)). As can be seen, the subject public key is the subject identity (i.e., the non-hierarchical foreign index pseudonym (nFIP))

(2) Then, the mobile device constructs the blind signature request message (BlndSigReq) which contains the non-hierarchical home request pseudonym (nHRP$_{i,r}$), the ID of the home server (ID$_h$), the blind hash (e), and a MAC generated on the previous fields using the home shared key (HSK$_i$). Then it sends the message to the home server, i.e., BlndSigReq=(ID$_h$ || nHRP$_{i,r}$|| e || MAC).

The home server receives the request and does the following.

(3) The home server uses the nHRP-Lnk to learn the non-hierarchical home index pseudonym. Then, to find the home shared key (HSK$_i$) which is used to verify the MAC, the home server multiplies the non-hierarchical home index pseudonym (elliptical curve public key) with its elliptical key private key (EPR$_h$). The result of this multiplication is the home shared key. This key is used to verify the MAC.

(4) If the MAC is successfully verified, the home server validates the index nonce. This validation guarantees the freshness of the non-hierarchical home request pseudonym (to protect

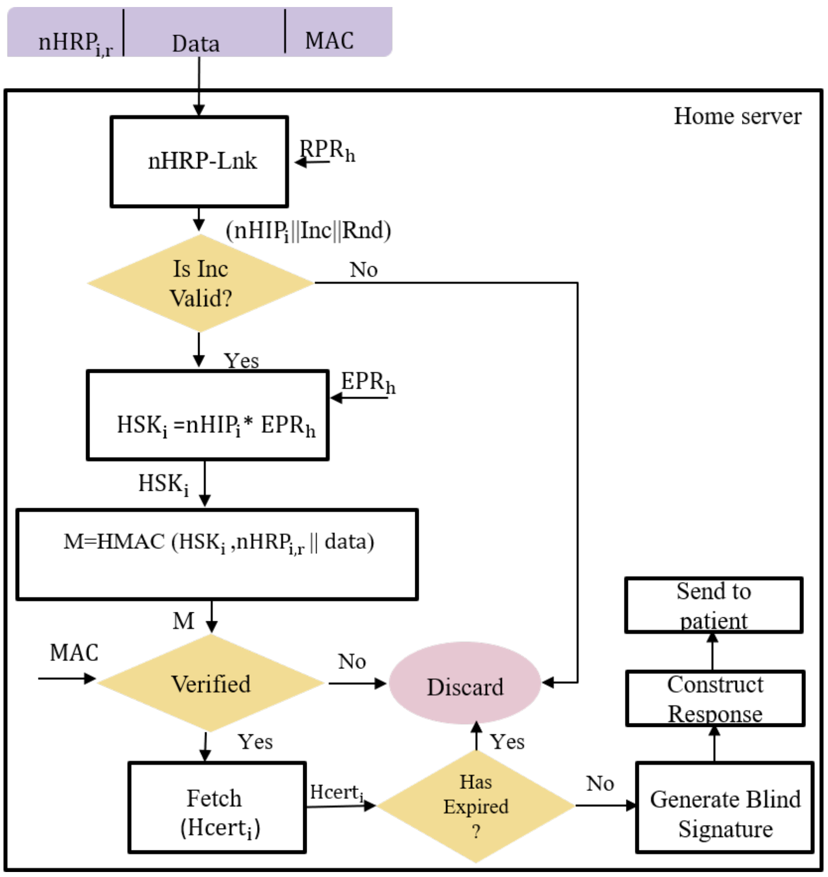


Figure 6.9: Authentication using non-hierarchical pseudonym

against replay attacks). It then checks that the HCert of the patient (HCert$_i$) has not expired. If both verifications are correct, the home server moves to the next step.

(5) The home server signs the blind hash (e) using the BlndSigGen algorithm.

(6) The home server constructs the blind signature response message (BlndSigRes), which contains the patient request pseudonym ID (nHRP$_{i,r}$), its Id (ID$_h$), and blind signature (S*). Then it generates the MAC on the message using (HSK$_i$) and sends the response to the patient's mobile device. i.e. BlndSigRes=(ID$_h$ || nHRP$_{i,r}$|| S* || MAC).

(7) The mobile device receives the BlndSigRes from the home server and does the following. The patient verifies the MAC associated with the message using his/her (HSK$_i$). The patient then extracts the blind signature (S*).

(8) Subsequently, the patient uses the Blind Signature Driven (BlndSigDrv) algorithm to deduce the unblind signature.

(9) Finally, the patient attaches the unblind signature in the signature field of the FCert. The certificate is now ready to be used. The patient constructs as many FCert as the number of foreign servers.

6.5 Requirement Analysis

In this section, the ADA protocols are analysed against the requirements set out in section 6.3.1.

- Undetectable: The HCP does not know which server the patient will choose as a home server. As one can see, the patient requests a home pseudonym certificate (HCert) from the HCP. The HCert is linked to the patient's home index pseudonym. Then, the patient selects any server to be the home server. The registration with the selected home server is done by sending the HCert along with the digital signature. The patient is not redirected to the HCP to do the registration. This is because the home server can verify both the HCP's signature on HCert and the patient's digital signature on the message. In addition, the HCP and the home server do not know the foreign servers that the patient is going to use. This is because the certificates (i.e., FCerts) which are used to authenticate the patient with the foreign servers are generated by the patient, and are blindly signed by the home server. Moreover, each foreign server does not re-direct the registration request from the patient to their home server. This is because each foreign server can register the patient by both verifying the home's proxy signature on the certificate and the patient's digital signature on the message.

- Inter-domain registration unlinkability: As one can see, the patient can use multiple data collection servers. The registration with each data collection server is done using a different certificate. For the home server, the patient uses HCert. For each foreign server, the patient uses a different FCert. In addition, through the generation of these certificates, the patient does not reveal any information about the servers which they will use. Thus linking different certificates to the same patient is a challenging task.

- One-time shows credential: The patient sends a certificate to the respective server only one time in the registration phase (as can be seen in both HSR and FSR protocols). For repeated authentication with the same server, the patient uses a fresh request pseudonym each time. The authentication of these pseudonyms is achieved by verifying the MAC associated with the message. For re-registering with the same server, the patient requests a new certificate.

- Intra-domain authentication unlinkability: The patient uses a new home request pseudo-nym for each request sent to the home server. These request pseudonyms are of two types: the hierarchical and non-hierarchical. The generation of the hierarchical request pseudonym is done by using a home shared key (HSK) known only to the patient and the home server. This key is difficult to discover because it relies on solving the EDCL problem. The generation of the non-hierarchical request pseudonym is done by using the RSA public key of the home server. The private key is hard to find, thus linking multiple requests to the same patient is computationally intensive. The same is true for hierarchical and non-hierarchical foreign request pseudonyms.

- Stateless request authentication: This is achieved by using the non-hierarchical pseudonym. A home server does not need to search the database for the verification key as it can be found as follows. The home server multiplies the non-hierarchical home index pseudonym (after the decryption process for the request pseudonym) with its ECC private key. This multiplication process results in the home shared key. This key is used to verify the request.

- Patient-centric: This is achieved by involving the patient in the generation of both types of credentials: certificates (FCerts) and pseudonym IDs.

6.6 Security Analysis

- The ADA is protected against an impersonation attack in the inter-domain anonymous registration protocol. Suppose that Eve learns Alice's certificate (i.e., HCert or FCert), Eve is attempting to play Alice's role and deceive a data collection server. Eve sends the certificate to the data collection server, and the server accepts the certificate from Eve. The data collection server finds that the certificate has been sent before by Alice, and in this case asks Eve to prove knowledge of the private key which corresponds to the public key stated in the certificate. Eve needs to prove to the data collection server that she is the owner of the certificate, so she needs to generate a digital signature using the private key which corresponds to the public key stated in the certificate. Eve fails to generate the digital signature as she does not have the private key. By design, Eve fails to impersonate Alice. In addition, both Alice's FCert and HCert, as shown in the protocols, are sent encrypted to the respective servers.

- The ADA is protected against an impersonation attack in the intra-domain anonymous authentication protocol. Suppose Eve wants to impersonate Alice by trying to generate Alice's home request pseudonym using the hierarchical method. As explained previously, to generate this type of pseudonym, Eve needs to know the home index pseudonym, index nonce, time, random nonce, and the secret key (HSK). The secret key is hard to obtain because generation of this key involves knowledge of Alice's private key. The knowledge of the private key lies in solving the elliptic curve discrete logarithm problem (ECDLP). Suppose, this time, Eve uses the non-hierarchical method to generate the pseudonym for Alice. This time, Eve needs to know the home index pseudonym (a public key), index nonce, time, random nonce, and the public key of the server. Suppose Eve has guessed all the parameters correctly, she can generate a non-hierarchical home request pseudonym, attach fake data and, generate the MAC using Eve's key. Eve then sends the request message to the server. The server receives and decrypts the request message and obtains the home index pseudonym. Next, it multiplies the home index pseudonym with the server elliptical curve private key to generate the shared key. The server then uses the shared key to verify the MAC of the message. Since the key is Eve's, the server fails to verify the message using the generated key and the server discards the message.

- The ADA is protected against the man in the middle attack in both protocols. Suppose a malicious server intercepts the messages from the patient and home server to make the patient believe it is the home server. The patient sends the HCert to the malicious server to register. The malicious server verifies both the HCP's signature on HCert and Alice's signature. The malicious server generates the shared key by multiplying its elliptical curve private key with the patient's elliptical curve public key (stated in the HCert). By using the generated shared key, the malicious server generates a MAC on response message and sends it to the patient. The patient receives the message from the malicious server and then verifies the MAC associated with the message using the home shared key. The verification step fails because the malicious server generates the MAC using its key, not a key which the patient used to generate the home shared key.

- ADA is protected against pseudonym forgery and brute force attacks as explained in Chapter 5.

- The ADA is also protected against replay attacks in both protocols. Suppose Eve tries to replay a home registration request message or a foreign registration request message. Eve learns the structure of the message and captures one of Alice's messages, replaying it later to disturb the server. When the server receives the message, it finds that it already has a record for Alice and discards the message. In the intra-domain anonymous authentication protocol, suppose Eve tries to replay a home authentication request message or a foreign authentication request message. Eve learns the structure of the message. Eve captures one of Alice's message and replays it later to disturb the server. When the server receives the message, it follows the steps described in the protocol. When the server decrypts the home request pseudonym and obtains the index nonce, it finds that the nonce is old and thus discards the message.

- The ADA is protected against repudiation attacks in both protocols. When the patient prepares a request message, s/he generates a digital signature on the message using his/her private key. The knowledge of the private key lies in solving the elliptic curve discrete logarithm problem (ECDLP).

- The ADA is protected against linkability attack. As explained in the protocol section each type of protocol involves using different pseudonyms. In the registration protocol, the patient uses a different index pseudonym and certificate for each server. In addition each request carried out by the patient with any server involves using fresh request pseudonyms, one for each request. Thus linking these request pseudonyms for the same patient is quite difficult, as explained in Chapter 5 in the degree of anonymity section.

6.7 Cost Evaluation

This section evaluates the performance of the proposed protocols in terms of computational cost imposed on various entities and the communication overhead.

6.7.1 Computational Cost

The computationally expensive cryptographic operations used in our protocols are asymmetric encryption/decryption and digital signature generation/verification. The inexpensive cryptographic operations include symmetric encryption/decryption and HMAC generation/verification. Below, we evaluate each of the protocols in terms of computational costs at each participating entity.

6.7.1.1 HSR Protocol

As described in section 6.4.3, during the execution of the HSR protocol:

- The mobile device performs the following operations upon the generation of a home registration request (hRegReq): one ECC signature generation to protect the authenticity of the request message, one RSA encryption to protect the confidentiality of the content of the request message (i.e., home pseudonym certificate and home index pseudonym), and one ECC shared key generation. The mobile device performs the following operations upon receiving a home registration response (hRegRes): one HMAC signature verification to verify the authenticity of the response message and one AES decryption to obtain the tag range.

- The home server performs the following operations upon receiving the home registration request (hRegReq): one RSA decryption to get the home pseudonym certificate and home index pseudonym, two ECC signature verifications to verify the authenticity of the request message and to verify the HCP signature on HCert, and one ECC shared key generation. Upon the generation of the home registration response (hRegRes), one HMAC generation to protect the authenticity of the response message and one AES encryption to protect the confidentiality of the tag range.

6.7.1.2 FSR Protocol

As described in section 6.4.3, during the execution of the FSR protocol:

- The mobile device performs the following operations upon the generation of a foreign registration request (fRegReq): one ECC signature generation to protect the authenticity of the request message, one RSA encryption to protect the confidentiality of the content of the request message (i.e., foreign pseudonym certificate and foreign index pseudonym), and one ECC shared key generation. The mobile device performs the following operations upon receiving foreign registration response (fRegRes): one HMAC signature verification to verify the authenticity of response message and one AES decryption to get the tag range.

- The foreign server performs the following operations upon receiving foreign registration request (fRegReq): one RSA decryption to obtain the foreign pseudonym certificate and foreign index pseudonym, two ECC signature verifications to verify the authenticity of the request message and to verify the HCP signature on FCert, and one ECC shared key generation. The server performs the following operations upon generation of the foreign registration response (fRegRes): one HMAC generation to protect the authenticity of the response message and one symmetric encryption to protect the confidentiality of the tag range.

6.7.1.3 Hierarchical FCertC Protocol

As described in section 6.4.4, during the execution of the FCertC protocol:

- The mobile device performs the following operations upon generation of a blind signature request (BlndSigReq): one RSA encryption to generate the foreign index pseudonym (FIP), one elliptical curve key pair generation, one AES encryption to generate the home request pseudonym, one hash, and one HMAC generation to protect the authenticity of the request message. Upon receiving a blind signature response (BlndSigRes): one HMAC signature verification is conducted and one operation is conducted to obtain the unblind signature from the blind one.

- The home server performs the following operations upon receiving the blind signature request: an (n) HMAC signature verification to learn the home shared key, one symmetric decryption to obtain the home index pseudonym, and one blind signature generation. Upon the generation of the blind signature response one HMAC generation is conducted to protect the authenticity of the response message.

6.7.1.4 Non-Hierarchical FCertC Protocol

As described in section 6.4.4, during the execution of the FCertC protocol:

- The mobile device performs the following operations upon generation a blind signature request (BlndSigReq): one RSA encryption to generate the non-hierarchical foreign index pseudonym (nFIP), one elliptical curve key pair generation, one asymmetric encryption to generate the non-hierarchical home request pseudonym, one hash, and one HMAC generation to protect the authenticity of the request message. Upon receiving a blind signature response (BlndSigRes): one HMAC signature verification is conducted and one operation to obtain the unblind signature from the blind one.

- The home server performs the following operations upon receiving the blind signature request: one asymmetric decryption to obtain the non-hierarchical home index pseudonym, one ECC multiplication to get the shared key, and one operation to generate the blind signature. Upon the generation of the blind signature response one HMAC generation is performed to protect the authenticity of the response message.

6.7.2 Communication Cost

The cost of communications is assessed by the number of bytes in each message. The communication cost for each protocol is acceptable in comparison with the UK average upload speed which is 3.7 Mbps [63].

Table 6.1: Length in bytes of the fields in the ADA protocol messages.

Filed	Size (Byte)
$\mathrm{HIP_i}$	32
$\mathrm{HRP_{i,r}}$	48
$\mathrm{HCert_i}$	256
$\mathrm{FIP_i^f}$	256
$\mathrm{FCert_i^f}$	256
$\mathrm{nHRP_{i,r}}$	256
$\mathrm{ID_f}$	4
tagNo	4
HMAC	64
e	32
$\mathrm{S^*}$	64

6.7.2.1 The HSR Protocol

With the HSR protocol, as described in section 6.4.3, the patient is registered and anonymously authenticated to the home server.

- The patient sends a request message, i.e., hRegReq $= (\mathrm{ID_h}||\mathrm{Me}||\sigma)$, where Me $= \mathrm{Enc}(\mathrm{M}, \mathrm{RPK_h})$, and $\mathrm{M} = (\mathrm{HIP_i}||\mathrm{HCert_i})$.
- The home server responds with, i.e., hRegRes $= (\mathrm{ID_h}||\mathrm{HIP_i}||\mathrm{Me}||\mathrm{MAC})$., where Me $= \mathrm{E}(\mathrm{M}, \mathrm{HSK_i})$. and $\mathrm{M} = (\mathrm{HIP_i}||\mathrm{TagRng}||\mathrm{Inc})$.

Thus, the communication overhead introduced by the HSR protocol from the patient to home server is as follows. The $\mathrm{HIP_i}$ size is 32 bytes, the $\mathrm{HCert_i}$ is 256 bytes. After that both $\mathrm{HIP_i}$ and $\mathrm{HCert_i}$ are signed using ESigGen with keys 256 bits long, so the σ is 64 bytes. Then, both the $\mathrm{HIP_i}$ and the $\mathrm{Hcert_i}$ are encrypted using RSA encryption algorithm with a key of 15360 bits long, so the length of the encrypted text (Me) is 1920 bytes. The ID of the message (hRegReq) is 4 bytes. The ID of the home server is 4 bytes. The total communication overheads introduced by the hRegReq is 1992 bytes, i.e., 4+4+1920+64. The communication overhead introduced by the HSR protocol from the home server to the patient is as follows. The TagRng is 4 bytes, and the index nonce (Inc) is 4 bytes. After that both index nonce (Inc) and the TagRng are encrypted using the AES algorithm with keys 256 bits long, so the length of the encrypted text (Me) is 48 bytes. Then, the HMAC is generated on the Me using key 256 bits long and the length is 64 bytes. The ID of the home server is 4 bytes, the ID of the message (hRegRes) is 4 bytes, and the $\mathrm{HIP_i}$ size is 32 bytes. The total communication overhead introduced by the hRegRes is 152 bytes, i.e., 4+4+32+48+64.

6.7.2.2 The FSR Protocol

With the FSR protocol, as described in section 6.4.3, the patient is registered and anonymously authenticated to a foreign server.

- The patient sends a request message, i.e., fRegReq $= (\mathrm{ID_f}||\mathrm{Me}||\sigma)$, where Me $= \mathrm{Enc}(\mathrm{M}, \mathrm{RPK_f})$, and $\mathrm{M} = (\mathrm{FIP_i^f}||\mathrm{FCert_i^f})$.

- The foreign server responses with, i.e., $\mathrm{fRegRes} = (\mathrm{ID_f}||\mathrm{FIP_i^f}||\mathrm{Me}||\mathrm{MAC})$., where $\mathrm{Me} = \mathrm{E}(\mathrm{M}, \mathrm{FSK_i^f})$. and $\mathrm{M} = (\mathrm{FIP_i^f}||\mathrm{TagRng}||\mathrm{Inc})$.

Thus, the communication overhead introduced by the FSR protocol from the patient to a foreign server is as follows. The $\mathrm{FIP_i^f}$ size is 256 bytes, the $\mathrm{FCert_i^f}$ is 256 bytes. After that both $\mathrm{FIP_i^f}$ and $\mathrm{FCert_i^f}$ are signed using ESigGen with keys 256 bits long, so the σ is 64 bytes. Then both $\mathrm{FIP_i^f}$ and $\mathrm{FCert_i^f}$ are encrypted using RSA encryption algorithm with a key of 15360 bits long, so the length of the encrypted text (Me) is 1920 bytes. The ID of the message (fRegReq) is 4 bytes and the ID of the foreign server is 4 bytes. The total communication overheads introduced by the fRegReq is 1992 bytes, i.e., 4+4+1920+64. The communication overhead introduced by the FSR protocol from the foreign server to the patient is as follows. The TagRng is 4 bytes, and the index nonce (Inc) is 4 bytes. After that both the index nonce (Inc) and TagRng are encrypted using AES algorithm which is keys 256 bits long, so the length of the encrypted text (Me) is 48 bytes. Then, HMAC is generated on the Me using a key 256 bits long and the length is 64 bytes. The ID of the foreign server is 4 bytes, the ID of the message (fRegRes) is 4 bytes and the $\mathrm{FIP_i^f}$ is 256. The total communication overhead introduced by the fRegRes is 376 bytes, i.e., 4+4+256+48+64.

6.7.2.3 The FCertC Protocol Using the Hierarchical Pseudonym

With the FCertC protocol, as described in section 6.4.4, the patient is authenticated to a home server using the hierarchical request pseudonym.

- The patient sends the following message to the home server: BlndSigReq=($\mathrm{ID_h}$ || $\mathrm{HRP_{i,r}}$|| TagNo || e || MAC).

- The home server responds to the patient with the following message, i.e., BlndSigRes=($\mathrm{ID_h}$ || $\mathrm{HRP_{i,r}}$|| $\mathrm{S^*}$ || MAC).

Thus, the communication overhead introduced by the FCertC protocol from the patient to the home server is as follows. The $\mathrm{ID_h}$ size is 4 bytes, the $\mathrm{HRP_{i,r}}$ is 48 bytes, the TagNo is 4 bytes, the e is 32 bytes, and the MAC is 64 bytes as well. The total communication overheads introduced by the BlndSigReq message is 152 bytes, i.e., 4+48+4+32+64. The communication overhead introduced by the FCertC protocol from the home server to the patient is as follows. The $\mathrm{ID_h}$ size is 4 bytes, the $\mathrm{HRP_{i,r}}$ is 48 bytes, the $\mathrm{S^*}$ is 64 bytes, and the MAC is 64 bytes as well. The total communication overheads introduced by the BlndSigRes message is 180 bytes, i.e., 4+48+64+64.

6.7.2.4 The FCertC Protocol using Non-Hierarchical Pseudonym

With the FCertC protocol, as described in section 6.4.4, the patient is authenticated to a home server using non-hierarchical request pseudonym.

- The patient sends the following message to the home server: BlndSigReq=(ID_h || $nHRP_{i,r}$|| e || MAC).

- The home server responds to the patient with the following message: BlndSigRes=(ID_h || $nHRP_{i,r}$|| S^* || MAC).

  Thus, the communication overhead introduced by the FCertC protocol at the patient to home server is as follows. The ID_h size is 4 bytes, the $nHRP_{i,r}$ is 256 bytes, the e is 32 bytes, and the MAC is 64 bytes. The total communication overheads introduced by the BlndSigReq message would be 356 bytes, i.e., 4+256+32+64. The communication overhead introduced by the FCertC protocol from the home server to the patient is as follows. The ID_h size is 4 bytes, the $nHRP_{i,r}$ is 256 bytes, the S^* is 64 bytes, and the MAC is 64 bytes as well. The total communication overheads introduced by the BlndSigRes message is 388 bytes, i.e., 4+256+64+64.

6.8 Performance Evaluation

The software which is used to implement the protocols is Java 2 Platform, Standard Edition (J2SE). Java is chosen because it supports a set of standard security interfaces. That is, it includes the Java Cryptography Extension (JCE). The JCE provides the implementation of several cryptographic primitives and key management services required in our protocols, including a secure random number generator, key management facilities, a message digest function (SHA256), block ciphers (AES and RSA), an elliptical curve cryptosystem (ECC), a blind signature, and a digital certificate. The databases which are used in the protocol are created using MySQL [64] as it is open-source. In this section, we describe the design and evaluation metrics of the experiment.

- Performance Evaluation Metrics: The performance metrics used for the ADA protocol (i.e., HSR, FSR, and FCert construction) evaluation are the average time a patient spends waiting in a queue, and the average response time of the system. We want these metrics to be within reason.

- Java Microbenchmark Harness (JMH): To measure the execution time for each protocol, we use a Java benchmarking tool called Java Microbenchmark Harness (JMH) [65].

- Hardware Architecture: To prototype the ADA protocols, a desktop computer running Windows 10 with a 1.99 GHz Intel Core i7 and 8GB of RAM is used. The timing results from the protocols execution presented here are based on this computer specifications.

- Queuing theory: The queuing theory [66] is used to predicate the performance of ADA protocols.

Table 6.2: Registration protocol execution time.

Operations	Key Size	Time (ms)
The patient executes the following algorithms		
Retrieve DataBase	-	0.293
ESigGen	256	1.1
doEPHD	256	1.7
RsaEncryption	15360	3.1
The server executes the following algorithms		
RsaDecryption	15360	729.2
CertVer	-	1.6
ESigVer	256	1.6
doEPHD	256	1.7
Store DataBase	-	3.4
HmacGen256	256	0.003
AesEncryption	256	0
The patient executes the following algorithms		
AesDecryption	256	0
HmacVer256	256	0.003
Store DataBase	-	3.4

Table 6.3: Hierarchical protocol execution time

Operations	Key Size	Time (ms)
Patient executes the following algorithms		
Retrieve DataBase	-	0.293
AesEncryption	256	0
RsaEncryption	2048	3.1
BlndMsgGen	-	0.594
HmacGen256	256	0.003
The home server executes the following algorithms		
FVK	256	21
AesDecryption	256	0
BlndSigGen	256	0
HmacGen256	256	0.003
The patient executes the following algorithms		
HmacVer256	256	0.003
BlndSigDrv	-	0
BlndSigVer	256	0.236
Store DataBase	-	3.4

Table 6.4: Non-hierarchical protocol execution time

Operations	Key Size	Time (ms)
The patient executes the following algorithms		
Retrieve DataBase	-	0.293
EKeyGen	256	0.293
RsaEncryption	2048	3.1
BlndMsgGen	-	0.594
HmacGen256	256	0.003
The home server executes the following algorithms		
RsaDecryption	2048	3.1
EMul	256	0.1
HmacVer256	256	0.003
BlndSigGen	256	0
HmacGen256	256	0.003
The patient executes the following algorithms		
HmacVer256	256	0.003
BlndSigDrv	-	0
BlndSigVer	256	0.236
Store DataBase	-	3.4

6.8.1 Queuing Theory

Queuing theory [66] is a mathematical study of waiting line systems (e.g. restaurant waiting lines). Queuing theory helps to predict the following: an average waiting time a user spends in the queue before being served, the total response time of the system, the number of users in the system, and the average utilization of the system. There are two models in queuing theory, the single sever model and the multi-server model. The single server queuing model (M/M/1) assumes there is only one server in the system. The multi-server queuing model (M/M/C), where C is the number of servers. To predict the performance metrics of our protocols, we applied queuing theory. We analyzed the performance of the protocols based on two models first when we have a single data collection server (M/M/1 single queuing model) handling patients' requests, and second when we have multiple servers (using M/M/C multiple servers model) handling the requests. Queuing theory uses mathematical formulas to theoretically analyze the performance metrics of a system. The mathematical formulas to predict the performance of a single server queuing model (M/M/1) are different from the formulas used to predict the performance of the multi-server queuing model (M/M/C).

Mathematical formulas for the single queue model :

- λ= mean arrival rate of patients (average number of patients arriving per unit of time).
- μ = mean service rate (average number of patients that can be served per unit of time).
- $\rho = \lambda/\mu$ = the average utilization of the system.
- L = $\lambda/(\mu - \lambda)$ = the average number of patients in the service system.
- L_Q = $\rho * L$ = the average number of patients waiting in line.
- W = $1/(\mu - \lambda)$ = the average time spent waiting in the system, including service time.
- W_Q = $\rho * W$ = the average time spent waiting in line.

Mathematical formulas for multi-server queuing model :

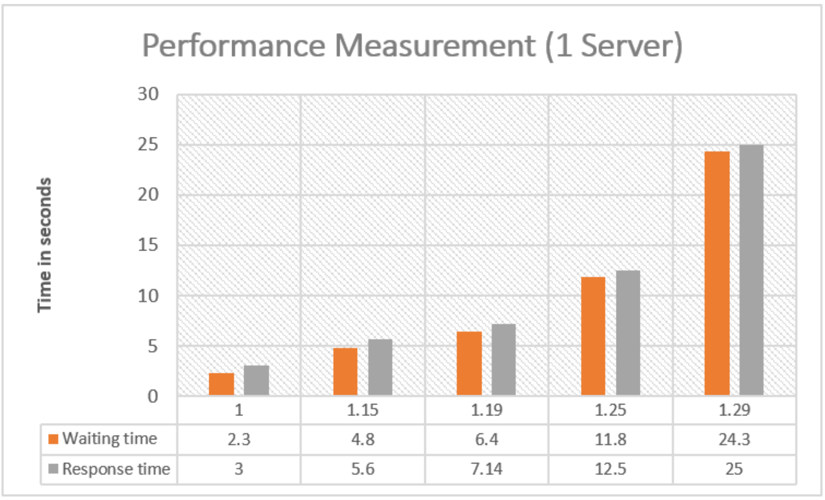
- s = the number of servers in the system.
- $\rho = \lambda/(s * \mu)$ = the average utilization of the system.
- P_0 = [$\sum_{n=0}^{s-1}(\lambda/\mu)/n! + ((\lambda/\mu)^s/s! * (1/1 - \rho))$ $]^{-1}$ = the probability that no patients are in the system.
- L_Q = $P_0(\lambda/\mu)^s\rho/s!(1 - \rho)^2$ = the average number of patients waiting in line.
- W_Q = L_Q/λ = the average time spent waiting in line.

- $W = W_Q + 1/\mu$ = the average time spent in the system, including service time.
- $L = \lambda * W$ = the average number of patients in the service system.

To theoretically analyze the performance of a protocol in terms of the waiting time for each request and the total response time of the system using queuing theory, we first need to determine the following inputs: the service time (m), the arrival rates of the requests (λ), and the service rate (μ). The service time (m) is the time the server needs to run the calculation for each request. In other words, it is the summation of the execution time of each method executed on the server side during the verification of the request and creation of the response. For example, the service time for the home registration request is the sum of the execution time of the methods executed on the server listed in Table 6.2 which is 0.75 seconds. The service rate μ can be calculated as (1/m). The arrival rates (λ) in case of single queuing model (M/M/1) should be less than μ. Otherwise, we will obtain negative values. In the multi-server queuing model (M/M/C), the arrival rates should be less than $\mu * s$, where s is the number of servers.

Table 6.5: Queue values for registration request

λ	ρ	L_Q	L	W_Q	W	P_0
Servers = 1						
1	75.1	2.278	3.03	2.278	3.03	24.8
1.15	86.4	5.524	6.389	4.804	5.556	13.53
1.19	89.47	7.605	8.500	6.391	7.143	10.526
1.25	93.98	14.685	15.625	11.748	12.500	6.015
1.29	96.99	31.280	32.250	24.248	25.00	3.0
Servers = 3						
1.5	37.6	0.07	1.2	0.049	0.80	31.8
1.9	47.6	0.19	1.6	0.10	0.85	22.8
2.5	62.7	0.65	2.5	0.261	1.013	13.2
2.9	72.7	1.42	3.6	0.45	1.24	8.4
3.5	82.7	5.6	8.2	1.6	2.3	3.1
3.9	97.7	41.5	44.4	10.6	11.4	0.5
Servers = 5						
3.5	52.6	0.172	2.803	0.049	0.801	6.96
3.9	58.6	0.312	3.244	0.080	0.832	5.03
4.5	67.7	0.715	4.1	0.15	0.91	2.98
4.9	73.7	1.2	4.9	0.25	1.0	2.04
5.5	82.7	2.91	7.04	0.53	1.3	1.04
6.5	97.7	40.91	45.79	6.294	7.045	0.092
Servers =7						
3.9	41.8	0.02	2.95	0.01	0.76	5.308
4.5	48.3	0.1	3.4	0.014	0.76	3.36
5.5	59.1	0.22	4.35	0.04	0.79	1.545
6.5	69.8	0.689	5.57	0.106	0.858	0.680
7.5	80.5	2.06	7.702	0.275	1.027	0.269
8.9	95	18.9	25.6	2.12	2.87	0.032
Servers =9						
5.5	45.9	0.024	4.16	0.004	0.75	1.59
6.5	54.3	0.085	4.97	0.013	0.765	0.746
7.5	62.65	0.246	5.885	0.033	0.78	0.34
8.5	71.0	0.638	7.03	0.075	0.8278	0.154
9.5	79.3	1.61	8.8	0.169	0.92	0.065
11	91.9	8.38	16.7	0.76	1.514	0.012



	1	1.15	1.19	1.25	1.29
🟧 Waiting time	2.3	4.8	6.4	11.8	24.3
⬜ Response time	3	5.6	7.14	12.5	25

Figure 6.10: Registration protocol performance using 1 server.

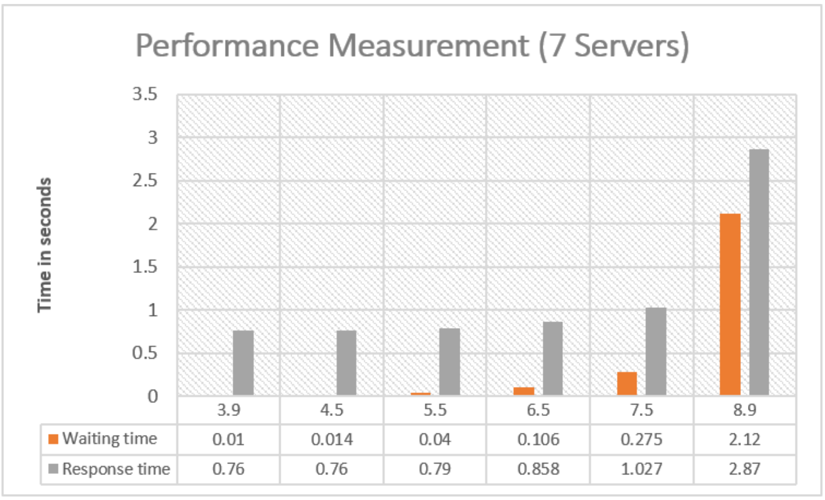
6.8.2 Registration Performance

In this section, we describe the performance of the home and foreign server registration protocols. To understand the performance of the protocols, we use queue modeling. In the single queue model the arrivals of patients' requests occur at lambda (λ) of exponential distribution. The service rate occurs at mu (μ) of the Poisson process. In the multi-server model, we select different numbers of servers (3, 5, 7, 9 servers).

- Single queue model: As one can see from the Table 6.5, we have calculated the performance of our system using the mathematical formulas of the single queuing model (M/M/1). In this case, the arrival rates should not exceed μ which is (1/0.75) = 1.33. We selected the arrival rates to range from 1 to 1.29. After conducting the calculations using the formulas above we found the following. The minimum waiting time and response time occurred when the arrival rate was 1 request per second. The maximum waiting time and response time arise when the arrival rates reach 1.29 requests per second. The overall average of the waiting time and response time of our system using one server are 9.9 seconds and 10.6 seconds respectively. Figure 6.10 shows the performance measurement of the registration protocol using one server.

- Multi-server queuing model (C=3): As one can see from Table 6.5, we have calculated the performance of our system using the mathematical formulas for the multi-server queuing model (M/M/C) for 3, 5, 7, and 9 servers. In the case of 3 servers, the arrival rates should not exceed the $\mu * s$ which is (1.33*3) = 3.9 requests per second. We selected the arrival rates to range from 1.5 to 3.9. The minimum waiting time and response time arise when the arrival rate is 1.5 requests per second. The maximum waiting time and response time arise when the arrival rate reaches 3.9 requests per second. The overall average of the waiting time and response time of our system using 3 servers are 2.2 seconds and 2.9 seconds respectively. As one can see, we enhance the system by 70% by using 3 servers.

- Multi-server queuing model (C=5): In the case of 5 servers, the arrival rates should not exceed the $\mu * s$ which is (1.33*5) = 6.65 requests per second. We selected the arrival rates to range



	3.9	4.5	5.5	6.5	7.5	8.9
■ Waiting time	0.01	0.014	0.04	0.106	0.275	2.12
■ Response time	0.76	0.76	0.79	0.858	1.027	2.87

Figure 6.11: Registration protocol performance using 7 servers.

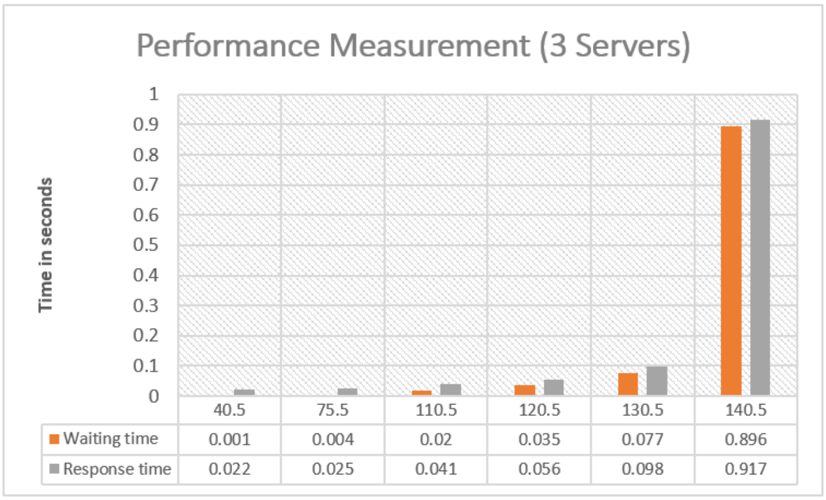
from 3.5 to 6.5. The minimum waiting time and response time arise when the arrival rate is 3.5 requests per second. The maximum waiting time and response time arise when the arrival rate reaches 6.5 requests per second. The overall average of the waiting time and response time of our system using 5 servers are 1.2 seconds and 1.9 seconds respectively. As one can see, we enhanced the system by 78% by using 5 servers.

- Multi-server queuing model (C=7): In the case of 7 servers, the arrival rates should not exceed the $\mu * s$ which is (1.33*7) = 9.31 requests per second. We selected the arrival rates to range from 3.9 to 8.9. The minimum waiting time and response time arise when the arrival rate is 3.9 requests per second. The maximum waiting time and response time arise when the arrival rate reaches 8.9 requests per second. The overall average of the waiting time and response time of our system using 7 servers are 0.43 seconds and 1.2 seconds respectively. Figure 6.11 shows the performance measurement of the registration protocol using 7 servers. We enhanced the system by 92% by using 7 servers.

- Multi-server queuing model (C=9): In the case of 9 servers, the arrival rates should not exceed the $\mu * s$ which is (1.33*9) = 11.97 requests per second. We selected the arrival rates to range from 5.5 to 11. The minimum waiting time and response time arise when the arrival rate is 5.5 requests per second. The maximum waiting time and response time arise when the arrival rate reaches 11 requests per second. The overall average of the waiting time and response time of our system using 9 servers are 0.18 seconds and 0.93 seconds respectively. We enhanced the system by 94% by using 9 servers.

6.8.2.1 The Hierarchical Authentication Performance

In this section, we describe the performance of session authentication using the hierarchical pseudonym in the FCertC protocol. The service time for the authentication using a hierarchical pseudonym is the summation of the execution time of the methods executed on the server listed in Table 6.3 which gives 0.021 seconds.



	40.5	75.5	110.5	120.5	130.5	140.5
■ Waiting time	0.001	0.004	0.02	0.035	0.077	0.896
■ Response time	0.022	0.025	0.041	0.056	0.098	0.917

Figure 6.12: Hierarchical protocol performance using 3 servers.

- Single queuing model: As one can see from Table 6.6, we have calculated the performance of our system using the mathematical formulas for the single queuing model (M/M/1), in which case the arrival rates should not exceed the μ which is (1/0.021) = 47.6 requests per second. We selected the arrival rates to range from 10.5 to 47. The minimum waiting time and response time arise when the arrival rate is 10.5 requests per second. The maximum waiting time and response time arise when the arrival rates reach 47 requests per second. The overall average of the waiting time and response time of our system using one server are 0.88 seconds and 0.95 seconds respectively.

- Multi-server queuing model (C=3): From the Table 6.6, we have calculated the performance of our system using the mathematical formulas for multiple queuing models (M/M/C). We have selected 3 and 7 servers. In the case of 3 servers, the arrival rates should not exceed $\mu * s$ which is (47.6*3) = 142.8 requests per second. We selected the arrival rates to range from 40.5 to 140.5. The minimum waiting time and response time arise when the arrival rates reach 100.5 requests per second. The maximum waiting time and response time arise when the arrival rate reaches 900.5 requests per second. The overall average of the waiting time and response time of our system using 3 servers are 0.17 second and 0.19 seconds respectively. The average response time is still higher than the actual service time (0.021 seconds). Figure 6.12 shows the performance of the authentication protocol using the hierarchical pseudonym.

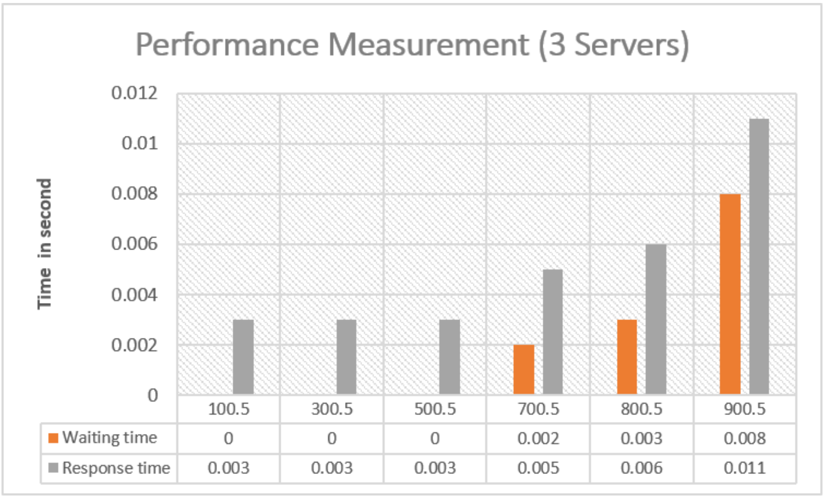
- Multi-server queuing model (C=7): In the case of 7 servers, the arrival rates should not exceed $\mu * s$ which is (47.6*7) = 333.2 requests per second. We selected the arrival rates to range from 100.5 to 330. The minimum, maximum, and average waiting time and response time are negligible.

6.8.2.2 Non-hierarchical Authentication Performance

In this section, we describe the performance of session authentication using the non-hierarchical pseudonym in FCertC protocol. The service time for the authentication using a non-hierarchical

Table 6.6: Queue values for hierarchical authentication

λ	ρ	L_Q	L	W_Q	W	P_0
Servers = 1						
10.5	22.2	0.065	0.29	0.007	0.027	77.75
15.5	32.8	0.161	0.5	0.010	0.032	67.2
20.5	43.4	0.33	0.8	0.016	0.037	56.56
30.5	64.6	1.18	1.83	0.04	0.1	35.4
40.5	85.8	5.187	6.045	0.128	0.149	14.2
47	99.6	234	235	4.9	5	0.424
Servers = 3						
40.5	28.6	0.025	0.88	0.001	0.022	42.13
75.5	53.3	0.313	1.912	0.004	0.025	18.73
110.5	78.03	2.18	4.53	0.020	0.041	6.32
120.5	85.1	4.2	6.734	0.035	0.056	3.934
130.5	92.2	10.1	12.8	0.077	0.098	1.904
140.5	99.2	125.85	128.83	0.9	0.917	0.174
Servers =7						
100.5	4.3	0	0.31	0	0	73.7
150.5	6.51	0	0.456	0	0.003	63.4
200.5	8.7	0	0.607	0	0.003	54.5
250.5	10.8	0	0.758	0	0.003	46.8
300.5	12.9	0	0.91	0	0.003	40.2
330	14.2	0	1	0	0.003	36.77



	100.5	300.5	500.5	700.5	800.5	900.5
🟧 Waiting time	0	0	0	0.002	0.003	0.008
⬜ Response time	0.003	0.003	0.003	0.005	0.006	0.011

Figure 6.13: Non-Hierarchical protocol performance using 3 servers.

pseudonym is the summation of the execution time of the methods executed on the server listed in Table 6.4 which gives 0.003 seconds.

- Single queuing model: Table 6.7 shows that we have calculated the performance of our system using the mathematical formulas for the single queuing model (M/M/1). In the case of the single queuing model (M/M/1), the arrival rates should not exceed μ which is (1/0.003) = 333.3 requests per second. We selected the arrival rates to range from 10.5 to 47. The minimum waiting time and response time arise when the arrival rate is 10.5 requests per second and the maximum waiting time and response time arise when the arrival rates reach 47 requests per second. The overall average of the waiting time and response times of our system using one server are 0.88 seconds and 0.95 seconds respectively.

- Multiple queues model (C=3): Table 6.7 shows the performance of our system using the mathematical formulas for multiple queuing models (M/M/C). We have selected 3 and 7 servers. In the case of 3 servers, the arrival rates should not exceed $\mu * s$ which is (333.3*3) = 999.9 requests per second. We selected the arrival rates to range from 100.5 to 900.5. The minimum waiting time and response time arise when the arrival rate is 100.5 requests per second and the maximum waiting time and response time arise when the arrival rate reaches 900.5 requests per second. The overall average of the waiting time and response time of our system using 3 servers are 0.17 seconds and 0.19 seconds respectively. Figure 6.12 shows the performance of the authentication protocol using the non-hierarchical pseudonym.

- Multiple queues model (C=7): In case of 7 servers, the arrival rates should not exceed $\mu * s$ which is (333.3*7) = 2333.1 requests per second. We selected the arrival rates to range from 1200 to 2333. The minimum, maximum, and average waiting time and response time are negligible. We can note that the service time of the non-hierarchical pseudonym authentication is less than the hierarchical one. Therefore, the number of requests per second is higher than with hierarchical authentication.

Table 6.7: Queue values for the non-hierarchical request

λ	ρ	L_Q	L	W_Q	W	P_0
Servers = 1						
95.5	28.7	0.12	0.402	0.001	0.004	71.3
100.5	30.2	0.130	0.432	0.001	0.004	69.8
150.5	45.1	0.372	0.823	0.002	0.005	54.8
200.5	60.2	0.91	1.51	0.005	0.008	39.84
250.5	75.2	2.27	3.025	0.009	0.012	24.84
300.5	90.2	8.260	9.2	0.027	50.03	9.8
325.5	97.5	38.182	39.16	0.125	0.128	2.5
Servers = 3						
100.5	10.2	0.00	0.302	0	0.003	73.9
300.5	30.1	0.030	0.93	0	0.003	40.28
500.5	50.1	0.238	1.74	0	0.003	21.01
700.5	70.1	1.154	3.3	0.002	0.005	9.54
800.5	80.1	2.6	5.004	0.003	0.006	5.6
900.5	90.1	7.4	10.11	0.008	0.011	2.5
Servers =7						
1200	6.1	0	0.43	0	0	65.14
1500	9.1	0	0.64	0	0	52.57
2200	13.4	0	0.943	0	0	38.94
2250	13.7	0	0.96	0	0	38.12
2333	14.3	0	1	0	0	36.7

6.9 Comparison with Related Works

We compare our ADA with the following works, as seen in Table 6.8, PseudoID [52], SPICE [58], Trustroam [40], and BLA [60]. These works are chosen because they are closer to our ADA. The PseudoID [52] is RSA-based system uses a new certificate for each service provider. Similar to PseudoID, SPICE [58] enables the user to create a new certificate based on one certificate issued by the identity provider, one certificate for each service provider. However, SPICE fails to provide anonymous linkage for multiple requests sent by the same

Table 6.8: Analysis of Relevant Works

Related Works	R1	R2	R3	R4	R5	R6
PseudoID	T	T	F	F	F	T
SPICE	T	T	F	F	F	T
Trust	T	T	F	F	F	T
BLA	T	T	F	F	F	T
ADA	T	T	T	T	T	T

patient to a service provider. The Trustroam [40] uses blockchain technology to store the certificates in distributed nodes instead of storing the certificates on the user side (mobile device). Trustroam enables authentication across different institutes using the same pseudonym ID leading to undesirable linkability across different domains. Similar to Trustroam, BLA [60] uses blockchain technology to conduct distributed authentication. However, BLA allows the use of different pseudonym IDs for each request given that one entity in each domain is able to link each pseudonym to the user's real ID. Table 6.8 summarizes the analysis of relevant works, where true (T) means the related work supports the requirement and false (F) means the related work does not support the requirement. In addition, R1 means undetectable, R2 means inter-domain registration unlinkability, R3 means one time registration certificate, R4 means intra-domain authentication unlinkability, R5 means stateless request authentication, and R6 means patient-centric.

The ADA solution is different from any other solution in the literature as it enables the patient from anonymous registration with any data collection server. For registering with a number of foreign servers, the patient generates FCert certificates signed blindly by the home server. For repeated authentication with the same data collection server, the patient does not need to send the same certificate or generate a new one, but s/he uses a request pseudonym. The request pseudonym is generated by the patient to facilitate anonymous authenticate per transaction. The request pseudonym is verified by the recipient server by two methods. First, if the request pseudonym is hierarchical, the server needs to search for the verification key using the tag number attached in the message. Otherwise, the server can find the key without searching the database by decrypting the request pseudonym using the server's RSA private key, then multiplying the result of the decryption with the server's ECC private key. The result of the multiplication is the verification key.

6.10 Chapter Summary

This chapter has presented novel anonymous distributed authentication protocols using certificates and pseudonym IDs. The certificates are of two types: the home pseudonym certificate (HCert) and foreign pseudonym certificate (FCert). The HCert is used to register with the home server. The FCert is used to register with a foreign server. Pseudonym IDs are used for repeated authentication with the same server. The non-hierarchical pseudonym has a faster authentication than the hierarchical one. This is because the verification key can be found without searching the database.

Chapter 7

Anonymous Authenticated Data Uploading and Forwarding (A2DUF) Protocol Suite

7.1 Chapter Introduction

This chapter describes an Anonymous Authenticated Data Uploading and Forwarding (A2DUF) protocol suite designed to support anonymous and pattern-hiding data uploading by a patient to the healthcare provider (HCP). After the patient does the registration and the authentications steps that are mentioned in Chapter 6, the patient can use the A2DUF suite that will be discussed in this chapter. The A2DUF suite consists of three protocols, a Data Uploading protocol, a Foreign2HomeServer Forwarding protocol and a Home2HCPServer Forwarding protocol. These protocols are, respectively, used for uploading data by a patient onto any of the patient's foreign servers, for forwarding the data by the patient's foreign servers to his/er home server, and for forwarding the data by the patient's home server to the healthcare provider. A swarming algorithm is embedded in the design of the Data Uploading protocol so as to hide the patient's uploading patterns. Anonymous data from the same patient but sent via the patient's multiple foreign servers is collected and anonymously linked by the patient's home server before being sent to the HCP. The HCP then links the patient's data to the patient's real ID. The performance evaluation of the uploading protocol is performed using queuing theory.

This chapter is organized as follows. Section 7.2 introduces the preliminary design of A2DUF that includes high-level ideas. Section 7.3 introduces the methods used in A2DUF. Section 7.4 describes the details of the A2DUF protocols. Section 7.5 analyzes the protocols against the design requirements. Section 7.6 analyzes the protocols against the security and ID privacy requirements. Section 7.7 evaluates the communication and computations costs of the proposed protocols. Section 7.8 describes the evaluation of the performance of the protocols using queuing theory. Section 7.9 compares the A2DUF to state-of-the-art anonymous distributed data collection solutions. Finally, in Section 7.10, a summary of the chapter is provided.

7.2 A2DUF Design Preliminaries

This section details the notations, assumptions, and requirements used in the design of the Anonymous Authenticated Data Uploading and Forwarding (A2DUF) protocol suite.

7.2.1 A2DUF Design Requirements

In addition to the design requirements identified in Chapters 4 and 6, the A2DUF should be designed to satisfy the following requirements.

(R1) Mutual anonymous authentication: As explained in Chapter 6, the patient should establish a mutual authentication with each server before uploading using FCerts.

(R2) Data confidentiality: Confidentiality protects data against unauthorized disclosure. No entity should be able to learn anything about the content of the patient's data except the HCP.

(R3) Data authenticity: Data authenticity assures that data is indeed from the claimed source and that it is exactly the same as that sent by the original sender. This means that each foreign server should be able to verify the authenticity of the uploaded data. The home server and the HCP should be able to verify the authenticity of the forwarded data. Early checking of authenticity is important to mitigate the burden of double verifications done by the HCP.

(R4) Uploading anonymity and unlinkability: This means that different uploads (one session may have a number of uploads) by the same patient are anonymous and unlinkable. Only the foreign server with which the patient is currently uploading should be able to link multiple uploads to the patient's records.

(R5) Unpredictable uploading pattern: This means that the patient's communication pattern (i.e., how many times and when the patient uploads to a server) should be uncertain. This is because the uploading pattern is an attribute that can be used to identify the patient.

(R6) Load distribution: This means that linking patient's records which are scattered across foreign servers should not be done by a single entity (i.e., the HCP).

(R7) On-event data uploading: In addition to periodic uploading, on-event data uploading should be supported. This to provide an on-time service for the patient in a critical health situation (e.g. breathing problems).

7.2.2 Threat Model

In addition to the threats discussed in the previous chapters, the A2DUF should be protected against the following threats.

- Replay attack: This is where an unauthorized entity records an upload message and reuses it later to do a malicious action.
- Data Tampering: This is when an unauthorized entity performs unauthorized modifications to the patient's data that are stored on a server or transmitted over a channel.
- Repudiation attack: This is where an entity takes part in a transaction then denies involvement in the transaction.

7.2.3 High-Level Ideas

In this section, we discuss the ideas we use to design the A2DUF to satisfy the design requirements.

- **Idea 1:** To satisfy the data confidentiality and authenticity requirements, we use two ideas:
  - Use MAC generation: The patient generates a MAC on an uploading message using a foreign shared key. The foreign server uses the key to verify the authenticity of the message. The key is only known to the patient and the foreign server.
  - Designing a double signing and encryption method: This method (as seen in Figure 7.1) is explained in Section 7.3. The purpose of this method is to prevent any foreign server or a home server from generating data on behalf of the patient. This method allows the home server to verify the authenticity of the patient's data forwarded by a number of foreign servers. In addition, it allows the HCP to verify the authenticity of the patient's data forwarded by the patient's home server.
- **Idea 2:** To satisfy the unlinkability and anonymity requirements, we use pseudonym IDs. The patient has different IDs for data collection servers. The patient uses different foreign index pseudonyms: one for each foreign server, one home index pseudonym for the home server, and the patient's real ID for the HCP. Moreover, for each uploading request with the same foreign server the patient uses a new request pseudonym that is generated based on the foreign index pseudonym. The same foreign server can link the uploading request to the patient's account (under the patient's foreign index pseudonym). Different foreign servers forward the patient's data to his/er home server. The home server can link the patient's data to the patient's account (under the patient's home index pseudonym). Finally, the home server forwards the patient's data to the HCP which links the patient's data to the patient's real ID.
- **Idea 3:** To satisfy unpredictable uploading pattern requirements, we propose a random uploading method called swarming. This method is explained in section 7.3.2.
- **Idea 4:** To support fair distribution of load, we support two levels of linkability. The first level is anonymous linkability. At this level, the home server of the patient should anonymously link the patient's data, which is scattered across different foreign servers, to the patient's home

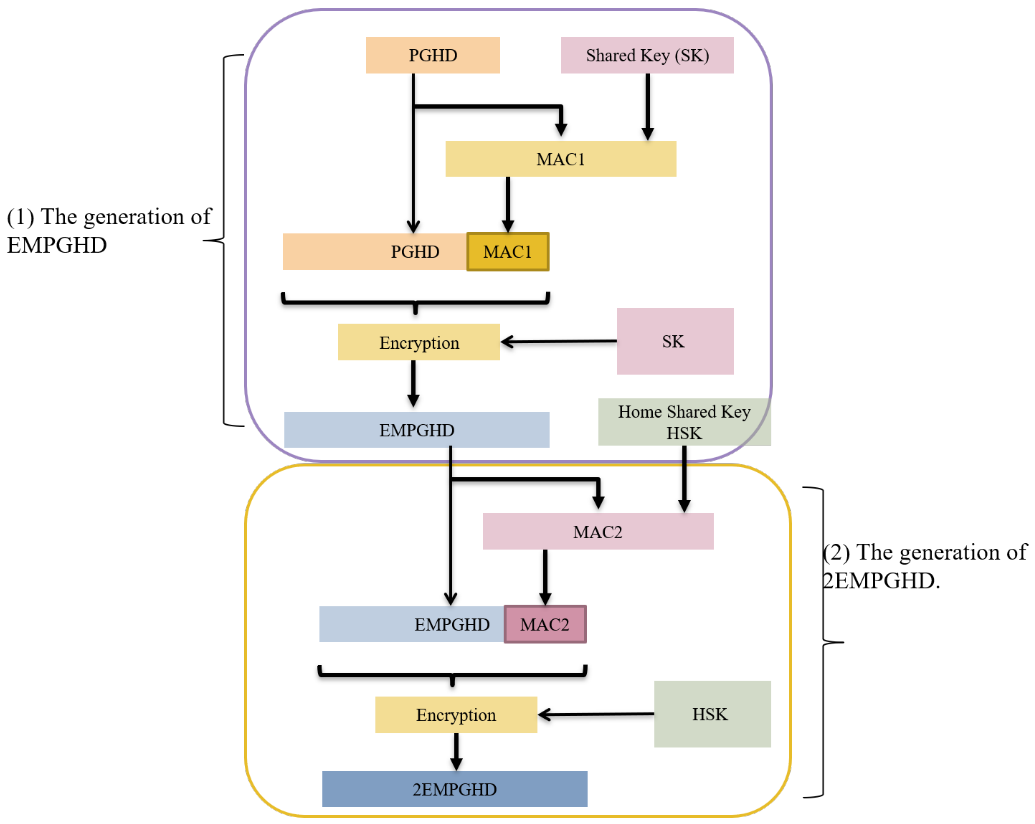


Figure 7.1: The sign then encrypt method

index pseudonym. The second level is to link the patient's data to the patient's real ID. This level is done by the HCP.

- **Idea 5:** To support on-event data uploading. The patient's health status is encoded into a request pseudonym. As explained in Chapter 5, the priority tag (Pr) is used to indicate how critical the patient's health status is. When a foreign server receives an upload message from the patient, the server first decrypts the request pseudonym to find the priority of the patient's data. If the priority tag is set, the foreign server sends a notification to the patient's home server which forwards the notification to the HCP.

7.2.4 Notation

The notation used throughout the method is summarized below. An n prefix indicates a non-hierarchical definition.

- HCP: The healthcare provider
- $\mathrm{PID_i}$: Patient Real ID for patient i.
- $\mathrm{MIP_i}$: Main Index Pseudonym for patient i.
- $\mathrm{HIP_i}$: Home Index Pseudonym for patient i.
- $\mathrm{FIP_i^f}$: Foreign Index Pseudonym for patient i with a foreign server f.

- $\mathrm{HRP}_{(i,r)}$: Home Request Pseudonym for a patient i for a request r.
- $\mathrm{FRP}^{f}_{(i,r)}$: Foreign Request Pseudonym for a patient i with a foreign server f.
- RPK_{e}: The RSA Public Key for entity e.
- RPR_{e} : The RSA Private Key for entity e.
- EPK_{e}: The Elliptical Curve Public Key for entity e.
- EPR_{e}: The Elliptical Curve Private Key for entity e.
- $\mathrm{SK}^{1}_{\mathrm{HCP}}$: The HCP First Secret Key.
- $\mathrm{SK}^{2}_{\mathrm{HCP}}$: The HCP Second Secret Key.
- SK_{i}: The Shared Key between patient i and HCP.
- HSK_{i}: The Home Shared Key for patient i with a home server.
- FSK^{f}_{i}: The Foreign Shared Key for patient i with a foreign server f.
- tPK^{f}_{i},tPR^{f}_{i}: Temporal Public and Private Keys generated by patient i with a foreign server f.
- $\mathrm{2EMPGHD}_{i}$: Double Encrypted and MAC Patient Generated Health Data.

7.2.5 Assumptions

- The communication between the patient and any data collection server takes place over the Internet using an anonymous communication channel.
- We assume that the mobile device has an agent that randomly selects a set of servers each day. No entity can discover how the mobile device selects these servers and the pattern of selection is assumed to be hidden.
- We assume that the combinatorial explosion problem will not occur.

7.3 A2DUF Methods

7.3.1 Double Signing and Encryption

To satisfy the data confidentiality and authenticity requirements we designed the double signing and encryption method. The method (as seen in Figure 7.1) can be explained as follows. The patient's data, which is uploaded to several foreign servers, has two forms. The first is Encrypted Message Authenticated Code and Patient Generated Health Data (EMPGHD), and the second form is Double Encrypted Message Authenticated Codes and Patient Generated Health Data (2EMPGHD). In EMPGHD, the patient generates a MAC on PGHD using the shared key (SK_i). This shared key is known to the patient and the HCP. Then, the patient encrypts both the PGHD and MAC with the same key. In 2EMPGHD, the patient generates the MAC on the EMPGHD using the home shared key (HSK_i) which is a key known to the patient and his/her home server. Then, the patient encrypts both the MAC and the EMPGHD using the home shared key. The result is the 2EMPGHD that will be uploaded onto different foreign servers. By using this method, the patient's data is kept confidential and authenticated through encryption and MAC generation, respectively. Only the HCP can learn the unencrypted form of the patient's data and verify the authenticity of the data. Only home server can verify the authenticity of the patient's data (i.e., EMPGHD) that is forwarded by foreign servers. By using this method, the HCP and the patient ensure that no entity can generate data on behalf of the patient and no entity can learn anything about the patient's data.

7.3.2 Swarming Method

To satisfy unpredictable uploading pattern requirement, we designed the swarming method. Swarming is a technique used by animals (e.g. birds) to defend themselves from predators. One can see a flock of birds twisting and moving from point to point in unpredictable ways to confuse those who want to eat them. It is a very complex behaviour. We borrow this idea from wildlife but use it in a way that is suited to our case study [4]. Swarming is used to hide the pattern of uploads and make the uploads random. Our Swarming method has three phases: creating uploading states, creating sub-states, and selecting from states.

7.3.2.1 Creating Uploading States

To create random uploads, we need to know in how many ways the patient's mobile device can divide the number of uploads (e.g. assume 3 2EMPGHD data packets) on a number of foreign servers (e.g. assume 2 servers). We use combinatorics to calculate the number of ways (states) to place n uploads into k servers. The combinatorics are as follows.

$$\binom{n+k-1}{k-1}$$

For example, suppose we have 3 uploads and 2 foreign servers in how many ways we divide the 3 uploads into 2 servers. Based on the above, we have n as the number of uploads and k as the number of servers. This means that we have 4 states as shown below.

$$\binom{3+2-1}{2-1} = \binom{4}{1} = 4$$

The first state is when the mobile device uploads 3 2EMPGHD packets to the first server (f_1) with no upload on the second server (f_2). This state can be represented as (3,0). The second state is when the mobile device uploads 3 2EMPGHD packets on the second server (f_2) with no upload on the first server. This state can be represented as (0,3). The third state is when the mobile device uploads 2 2EMPGHD packets on the first server (f_1) and 1 2EMPGHD packet on the second server (f_2). This state can be represented as (2,1). The fourth state is when the mobile device decides to upload 1 2EMPGHD packet on the first server (f_1) and 2 2EMPGHD packets on the second server (f_2). This state can be represented as (3,0). These states are shown in Figure 7.2. It should be noted that when the number of servers and number of packets increase, we will have a large set of states. It will be challenging for an external or internal entity to guess the next state out of many states.

We shall continue on the following example until the end of this section. Suppose we have three foreign servers and sixteen 2EMPGHD packets, this means we have 153 states. The mobile device calculates all the 153 states. One of the states could be (5,5,6), which means uploading 5 2EMPGHD packets to the f_1 server , 5 2EMPGHD packets to the f_2 server, and 6 2EMPGHD packets to the server f_3. For a higher protection level, the mobile device should not upload 5 sequential data packets to the same server at a time. For example, the mobile device should not upload at time (t_1) 1 packet to the f_1 server then at (t_2) 1 packet to the f_1 server and at (t_3) 1 packet to the f_1 server and so on. For a high level of protection, the 5 uploads should be divided among a number of rounds. For illustration, in the first round the mobile device uploads to the f_1 server, in the second round the mobile device uploads to the f_2 server and so on. Therefore, we need to create sub-states for each state.

7.3.2.2 Creating Sub-states

After the mobile device selects one state randomly (i.e, (5,5,6)), it creates sub-states out of this state. First, the mobile device selects the number of rounds randomly for each server. Suppose the mobile device selects three rounds. This means that the mobile device will finish uploading the 5 data packets on the first server (f_1) across three rounds, and the same for the other servers. Second, the mobile device needs to know how many ways (sub-states) to distribute the number of data packets for each server on three rounds. For the first server, the mobile device has to upload 5 data packets. By using the combinations below there are 21 sub-states for the first server. The mobile device lists these sub-states and randomly selects one sub-state. For example, the mobile device may randomly select (1,2,2) as a sub-state. This means that at the first round uploads 1 data packet, at the second round uploads 2 data packets, at the third round uploads 2 data packets onto the first server.

$$\binom{5+3-1}{3-1} = \binom{7}{2} = 21$$

After finishing calculating and listing the sub-states for the first server, the mobile device calculates sub-states for the second and third servers. Let's assume that the mobile selects the (2,2,1) sub-state for server f_2, and (4,1,1) for server f_3.

Now, the mobile device has three states, one for each server. For the first server f_1 the

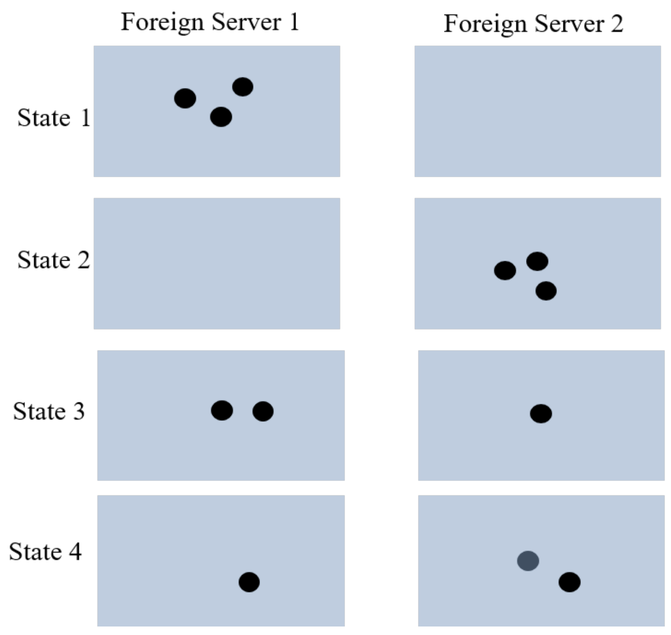


Figure 7.2: Different uploading states

selected sub-state was (1,2,2), therefore, the states of f_1 are {(1,0,0), (2,0,0), (2,0,0)}. For the second server f_2 the selected sub-state was (2,2,1), therefore, the states of f_2 are {(0,2,0), (0,2,0), (0,1,0)}. For the third server f_3 the selected sub-state was (4,1,1), therefore, the states of f_3 are {(0,0,4), (0,0,1), (0,0,1)}.

7.3.2.3 Selecting from States

Next, the mobile device selects one state randomly from the states, i.e., {(1,0,0), (2,0,0), (2,0,0), (0,2,0), (0,2,0), (0,1,0), (0,0,4), (0,0,1), (0,0,1)}. Suppose the mobile device selects the state (1,0,0). This means that for this round it uploads 1 data packet (i.e., 2EMPGHD) to the first server. Then, the mobile device selects another random state, which could be (0,0,4). This means that it uploads 4 data packets on the third server. The mobile device keeps selecting randomly from the states until they finish.

It should be noted that by allowing each patient's mobile device to upload to different servers each time using a random strategy, we are creating a swarming mechanism. This is because different patients are swarming unpredictably among these servers. This swarming behaviour makes it very hard for any entity to understand a given patient's behaviour and to know where in the future s/he will send the data.

7.4 A2DUF Protocol Suite

The A2DUF protocol suite consists of three protocols: (1) a Data Uploading protocol which is used between a patient and the patient's foreign servers, (2) a Foreign2HomeServer Forwarding protocol which is used between a patient's foreign server and the patient's home server, and (3) a Home2HCPServer Forwarding protocol which is used between a patient's home server and the healthcare provider.

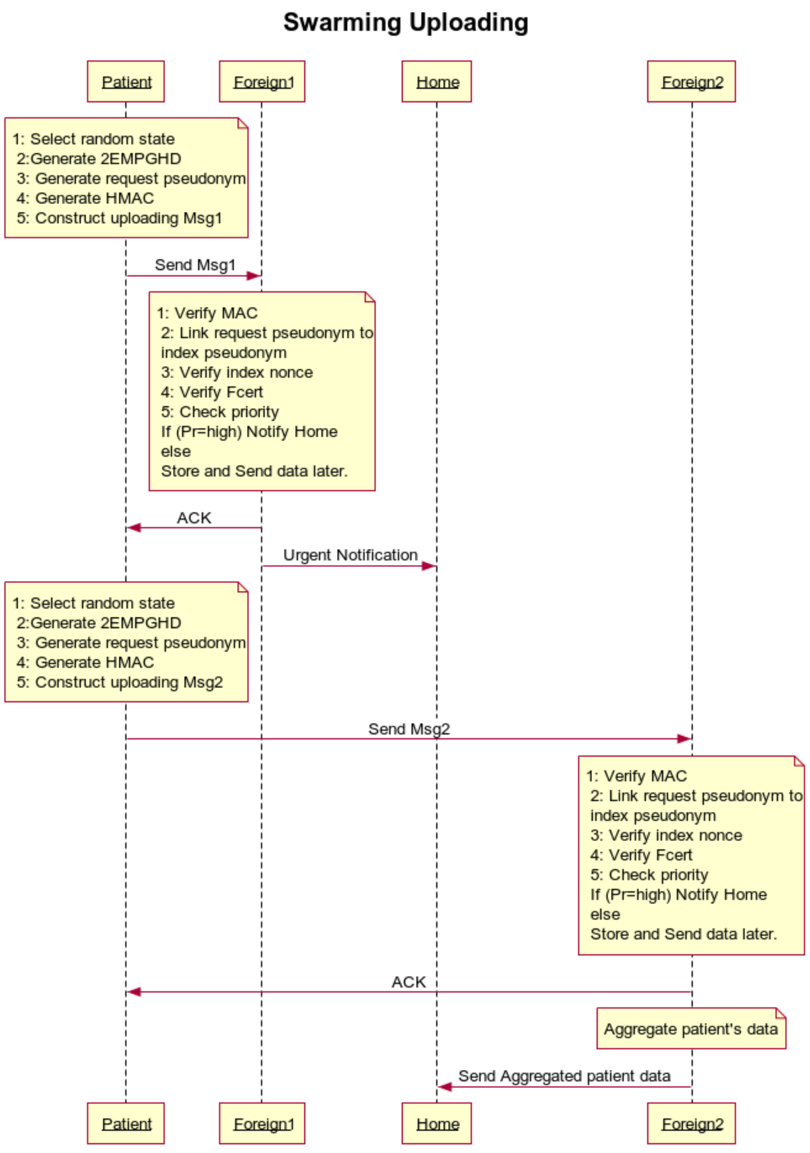


Figure 7.3: Swarming uploading

7.4.1 Data Uploading Protocol

This protocol, as seen in Figure 7.3, is executed between the patient and a number of foreign servers. The purpose of this protocol is to allow the patient to upload his/her data to any foreign server. The uploading protocol starts as follows.

(1) The mobile device selects a state as explained in section 7.3.2. Assume the selected state is (1,0,0), which means in this round there is one upload to the first server and zero uploads on other servers. After that, the mobile device generates a form of data called 2EMPGHD as explained in section 7.5.

(2) Then, the patient's mobile device generates a foreign request pseudonym ($FRP^{f}_{i,r}$) using the FRPs-Gen algorithm. Then, it selects a tag number (TagNo) from the tag ranges. The priority of the patient's data is encapsulated in the foreign request pseudonym.

(3) The mobile device constructs an uploading request (UpReq) message. This message contains the ID of the foreign server (ID_f), the ID of the patient ($FRP^{f}_{i,r}$), the tag number (TagNo), and the patient's data ($2EMPGHD_i$).

(4) The mobile device generates a MAC on the message using the foreign shared key (FSK_i^f).

(5) The mobile device sends the uploading request (UpReq) message to the foreign server.

$\mathbf{Patient}_i \rightarrow \mathbf{ID_f}$**:(**$\mathbf{ID}_f$ **||**$\mathbf{FRP^f_{i,r}}$ **||TagNo ||2EMPGHD ||MAC**).

The foreign server receives the request message and performs the following.

(6) It uses the tag number (TagNo) and FVK method to search for the verification key (FSK_i^f) to verify the MAC. If the MAC is successfully verified, it moves to step (7), otherwise, it discards the message.

(7) The foreign server uses the key as an input to the FRPs-Lnk algorithm. This is to find the foreign index pseudonym (FIP_i^f).

(8) After finding the index pseudonym for the patient, the foreign server validates the index nonce. This validation guarantees the freshness of the foreign request pseudonym. It then checks that the foreign pseudonym certificate (FCert) of the patient has not expired (e.g. within 24 hours). If both verifications are correct, the foreign server stores the data.

(9) Next, the foreign server checks the priority tag (PR). If the priority tag is set, it sends a notification to the patient's home server. Otherwise, it stores the patient's 2EMPGHD and, after a period, it aggregates all the patients' data and sends it to the home servers.

The mobile device has to select another state. Suppose the next state is (0,2.0), this means that the mobile device will make 2 uploads to the second server and zero to the other servers. This time the patient may use the non-hierarchical pseudonym. The uploading process is as follows.

(1) The patient's mobile device generates the 2EMPGHD using a double signing and encryption method.

(2) Then, the patient's mobile device generates a non-hierarchical foreign request pseudonym ($nFRP^f_{i,r}$) using the nFRPs-Gen algorithm.

(3) Subsequently, it constructs an uploading request (UpReq) message. This message contains the foreign server ID (ID_f), the patient's ID ($nFRP^f_{i,r}$), and the patient's data ($2EMPGHD_i$).

(4) Then, the mobile device generates a MAC on the message by using the foreign shared key (FSK_i^f).

(5) The mobile device then sends an uploading request (UpReq) message to the foreign server.

$\mathbf{Patient}_i \rightarrow \mathbf{ID_f}$**:(**$\mathbf{ID}_f$ **||**$\mathbf{nFRP^f_{i,r}}$ **||2EMPGHD ||MAC**).

The foreign server receives the request message and performs the following.

(6) It uses its RSA private key (RPR_f) as an input to the nFRP-Lnk algorithm to decrypt the ($nFRP^f_{i,r}$) and find the temporal elliptical curve public key (tPK^f_i) of the patient and other information related to the patient (i.e., nonce, priority of the data). The foreign shared key can be found by multiplying the (RPR_f) with the (tPK^f_i). The result is the foreign shared key which is used to verify the MAC.

(7) The foreign server next validates the index nonce. This validation is to guarantee the freshness of the foreign request pseudonym (i.e., protect against replay attack). It then checks that the foreign pseudonym certificate (FCert) of the patient has not expired (i.e., within 24 hours). If both verifications are correct, the foreign server sends an acknowledgement to the patient.

(8) Next, the foreign server checks the priority tag (PR). If the priority tag is set, it sends a notification to the patient's home server. Otherwise, it stores the patient's 2EMPGHD.

Since the selected state was (0,2.0), the mobile device uploads the second upload to the second server. After finishing uploading, the mobile device selects another state. The process of selecting states should be continued until the mobile device finishes all the uploads expected per day.

7.4.2 Foreign2HomeServer Forwarding Protocol

This protocol is executed between data collection servers. It allows each data collection server which worked as a patient's foreign server to forward the patient's data, after aggregating them, to the patient's home server. The protocol can be described as follows.

(1) The foreign server concatenates the patient's data along with the patient foreign index pseudonym in a data structure (i.e., FIP^f_i || $2EMPGHD_i, 1$||...|| $2EMPGHD_{i,n}$). It should be noted that data structures for different patients will be aggregated in one packet to save bandwidth.

(2) Next, the foreign server constructs a forward request (FrwReq) message which contains the ID for both servers, and the patients' aggregated data structures (AGGD). The server then generates a MAC on the message using the shared key. The server then sends the message to the patient's home server.

$$\mathbf{ID}_f \rightarrow \mathbf{ID_h}\mathbf{:(ID}_f \; \| \mathbf{ID}_h \; \| \mathbf{AGGD}_i \; \| \mathbf{MAC).}$$

(3) The FrwReq message arrives to the home server which uses the foreign server ID to find its certificate and then extract the ECC public key. By using the public key, the home server can generate the shared key.

(4) After finding the key, the home server verifies the MAC. If the MAC was successfully verified, the home server can extract the data structure for each patient (i.e., $\mathrm{FIP}_{\mathrm{i}}^{\mathrm{f}}$|| $2\mathrm{EMPGHD}_{i,1}$|| $2\mathrm{EMPGHD}_{i,2}$ ||...|| $2\mathrm{EMPGHD}_{i,n}$).

(5) Then for each data structure, the home server recovers the patient's home index pseudonym (i.e., HIP_i) from the foreign index pseudonym (i.e., $\mathrm{FIP}_{\mathrm{i}}^{\mathrm{f}}$) by using the FIP-Lnk algorithm.

(6) The home server then uses the home index pseudonym (HIP_i) to find the home shared key HSK_i to decrypt the patient's data (i.e., 2EMPGHD) and to obtain the encrypted data for the patient (i.e., EMPGHD) and the associated MAC (MAC2).

(7) Next, the home server uses the home shared key HSK_i to verify the second MAC (MAC2). If successfully verified, it stores the patient's data (EMPGHD) in the database until it is requested by the HCP.

(8) If the foreign index pseudonym is non-hierarchical, then the home server recovers the patient's non-hierarchical home index pseudonym (i.e., nHIP_i) from the foreign index pseudonym ($\mathrm{nFIP}_{\mathrm{i}}^{\mathrm{f}}$) by using the nFIP-Lnk algorithm.

(9) Then, the home server uses the non-hierarchical home index pseudonym (nHIP_i) to find the home shared key (HSK_i). This is done by multiplying the (nHIP_i) with the ECC private key of the home server (EPR_h). The home shared key is used to decrypt the patient's data (i.e., 2EMPGHD) to get the EMPGHD and the associated MAC (MAC2).

(10) The home server uses the home shared key (HSK_i) to verify the second MAC (MAC2). If successfully verified, it stores the patient's data (EMPGHD) in the database until it is requested by the healthcare provider.

7.4.3 Home2HCPServer Forwarding Protocol

This protocol is executed between a data collection server and the healthcare provider (HCP). This protocol allows the data collection server which worked as a home server for some patients to send the patients' data to the final destination (HCP).

(1) Each data collection server retrieves a patient's home index pseudonym, and the patient's data (i.e., EMPGHD). The server concatenates the patient's data along with the patient's home index pseudonym in a data structure (i.e. HIP_i || $\mathrm{EMPGHD}_{i,1}$|| $\mathrm{EMPGHD}_{i,2}$ ||...|| $\mathrm{EMPGHD}_{i,n}$). Multiple data structures will be aggregated in one packet for multiple patients.

(2) Next, the server constructs a home forward request (HFrwReq) message which contains the ID of both servers and the aggregated data structures of the patients (AGGD). The server then generates a MAC on the message using the shared key and sends the message to the HCP.

$\mathbf{ID}_h \rightarrow$**HCP:(**$\mathbf{ID}_h$ **||**$\mathbf{ID}_{HCP}$ **||AGGD ||MAC)**.

(3) After obtaining the key, the HCP verifies the MAC. If the MAC was successfully verified, the HCP extracts the data structure for each patient (i.e., HIP$_i$ || EMPGHD$_{i,1}$|| EMPGHD$_{i,2}$ ||...|| EMPGHD$_{i,n}$).

(4) Then for each data structure, the HCP recovers the patient's real ID (i.e., PID$_i$) from the home index pseudonym (HIP$_i$) by using the HIP-Lnk algorithm.

(5) Then, the HCP uses the patient's real ID (PID$_i$) to find the shared key SK$_i$ to decrypt the patient's data (i.e., EMPGHD) and to obtain the unencrypted patient's data (i.e., PGHD) and the associated MAC (MAC1).

(6) Then, the HCP uses the shared key SK$_i$ to verify the first MAC (MAC1). If successfully verified, The HCP stores the patient's data (PGHD) in the database.

7.5 Requirement Analysis

In this section, the A2DUF is analyzed against the requirements set out in section 7.3.

- The A2DUF supports mutual anonymous authentication as explained in Chapter 6.
- The A2DUF supports data confidentiality: In the data uploading protocol, the patient's data that is uploaded on any foreign server is double encrypted using two keys. One of the encryption key is known only to the patient and the HCP. Therefore, no entity can decrypt and learn the patient's data.
- The A2DUF supports data authenticity: In the data uploading protocol, the patient generates a MAC on the upload message before sending it to a foreign server. By using the MAC, the foreign server guarantees that the message has not been altered from the point at which it was dispatched. The key used to generate the MAC is generated from the private key of the patient. This guarantees that no other entity can generate the key and the message is from the claimed patient. In the Foreign2HomeServer forwarding protocol, the patient's data is in the form of 2EMPGHD, and is double signed and encrypted. When the patient's data arrives at the home server, it can verify the integrity of this data using a home shared key which is only known to the patient and the home server. In the Home2HCPServer forwarding protocol, the patient's data is in the form of EMPGHD, and is signed and encrypted. When the patient's data arrives at the HCP, it can verify the integrity of this data using a shared key which is only known to the patient and the HCP.
- The A2DUF supports anonymous and unlinkable uploading: For each upload with any foreign server, a patient uses a new foreign request pseudonym generated using either hierarchical or non-hierarchical methods. Linking different request pseudonyms to different patients is a challenging task as explained in Chapter 5.

- The A2DUF supports unpredictable uploading patterns: This is achieved by using the swarming technique which is explained in section 7.3.2.

- The A2DUF supports on-event data uploading by encapsulating the priority of the data inside the pseudonym: When a foreign server receives an upload message from the patient, it decrypts the request pseudonym. After the decryption process, the foreign server checks the priority tag; If set, then it sends a notification message to the patient's home sever.

- The A2DUF supports load distribution: Each home server is responsible for linking the patient's data, scattered across multiple foreign servers, to the patient's home account. The home server is responsible for verifying the authenticity of the data (i.e., 2EMPGHD form) before sending it to the HCP. This reduces the overhead on the HCP which only verifies the authenticity of the EMPGHD form.

7.6 Threat Analysis

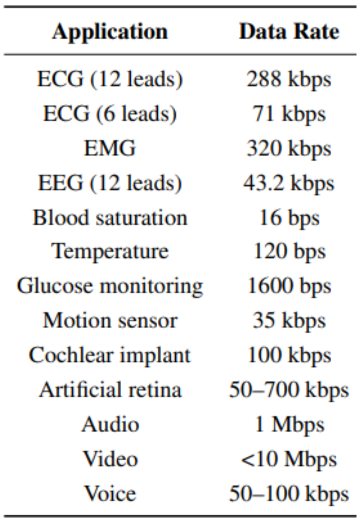
In this section, the A2DUF is analysed against threats set out in section 7.4. Here, Alice is an authorised patient. Eve is an adversary.

- A2DUF is protected against a data forgery attack launched by an internal entity: Suppose a home server tries to generate data for Alice. However, the home server cannot generate data for Alice. This is because the home server needs to obtain Alice's shared key (SK) (i.e., the key between Alice and the healthcare provider server) to generate the EMPGHD. Further if the foreign server tries to generate data for Alice, the foreign server needs to obtain two keys. The first key is the SK used to generate the EMPGHD and the second is the home shared key (HSK) (i.e., the one between the patient and home server) to generate the 2EMPGHD but the foreign server cannot generate data for Alice.

- A2DUF is protected against data forgery attack launched by an external entity: Suppose Eve has captured Alice's uploading message; as explained previously the message contains the following fields: Alice's ID (i.e., request pseudonym), Alice's data (i.e. EMPGHD/2EMPGHD), the tag number (if the pseudonym is hierarchical), and the MAC generated on the message fields. Suppose Eve can recognise the length of the uploading message and learn how to extract message fields. Eve changes the tag number (adding her own), uses Alice's request pseudonym and changes the data to her own data. After that, Eve generates the MAC on the new message. Eve sends the message to the data collection server, which uses the tag number to calculate the first and last index and uses these boundaries (i.e., first and last indexes) to search for the verification key. When the server finds the key to verify the message, it uses the same key to decrypt the request pseudonym. Since Eve's key is not the key that Alice used to generate the request pseudonym, the server will not decrypt the request pseudonym correctly. Therefore, the server discards the message. In a similar scenario, but this time with Alice using the request pseudonym which is generated by the non-hierarchical method, Eve performs the same process of generating the MAC on the request pseudonym and her data.

Table 7.1: Length in bytes of the fields in the A2DUF protocol messages.

Filed	Size (Byte)
$FRP^{f}_{i,r}$	272
$nFRP^{f}_{i,r}$	256
ID_f	4
tagNo	4
HMAC	64
2EMPGHD	variable
AGG	variable



Application	Data Rate
ECG (12 leads)	288 kbps
ECG (6 leads)	71 kbps
EMG	320 kbps
EEG (12 leads)	43.2 kbps
Blood saturation	16 bps
Temperature	120 bps
Glucose monitoring	1600 bps
Motion sensor	35 kbps
Cochlear implant	100 kbps
Artificial retina	50–700 kbps
Audio	1 Mbps
Video	<10 Mbps
Voice	50–100 kbps

Figure 7.4: The data rates of wearable sensors [48]

This time the foreign server decrypts the request pseudonym, extracts the index pseudonym, and multiplies the index pseudonym after decoding it (recall this is a public key) with the foreign server's elliptical curve private key. This process results in a shared key. The key will not verify the MAC associated with the message so the server discards the message.

- A2DUF is protected against replay attack: Suppose Eve tries to replay uploading messages to disturb the server. Eve forwards Alice's old uploading messages to a foreign server. After the server performs all the verifications, the server verifies the freshness of the index nonce, and discovers that the messages sent by Eve are old and so discards the messages.

- A2DUF is protected against repudiation attacks: Messages exchanged between Alice and a data collection server contain a MAC generated by Alice's shared key. As explained previously, generating the shared key involves using both parties' private keys. The knowledge of the private key requires solving the elliptic curve discrete logarithm problem (ECDLP), which has been proven to not be computationally feasible. Thus, it is very hard for Eve to generate a message on behalf of Alice. So Alice can not claim that an entity has generated a message of behalf of her.

7.7 Cost Evaluation

This section evaluates the cost of the proposed protocols in terms of computational and communication overhead.

7.7.1 Computational Costs

The computationally expensive cryptographic operations used in our protocols are asymmetric encryption/decryption and digital signature generation/verification. The inexpensive cryptographic operations include symmetric encryption/decryption and HMAC generation/verification. Below, we evaluate each of the protocols in terms of computational costs at each participating entity.

7.7.1.1 Data Uploading Protocol

As described in section 7.4.1, the cryptographic operations which are used during the execution of the data uploading protocol using hierarchical pseudonym are as follows.

- The mobile device performs the following operations upon the generation of an uploading request (UpReq): two HMACs generation and two AES encryptions to protect the data authenticity and confidentiality of the patient's data, one AES encryption to generate the pseudonym, and one HMAC generation to protect the authenticity of the request message. Upon receiving an uploading response (UpRes): the mobile device generates one HMAC signature verification to verify the authenticity of response message.

- The foreign server performs the following operations upon receiving the uploading request (UpReq): (n) HMAC verifications to verify the authenticity of the message, and one AES decryption to obtain the foreign index pseudonym. Upon the generation of the uploading response (UpRes): the foreign server generates one HMAC signature to protect the authenticity of the response message.

The cryptographic operations during the execution of the data uploading protocol using a non-hierarchical pseudonym are as follows.

- The mobile device performs the following operations upon the generation of an uploading request (UpReq): two HMAC signatures and AES encryptions are generated to protect the data authenticity and confidentiality of the patient's data, one RSA encryption to generate the pseudonym, and one HMAC signature to protect the authenticity of the request message. Upon receiving an uploading response (UpRes): the mobile device performs one HMAC signature verification to verify the authenticity of response message.

- The foreign server performs the following operations upon receiving the uploading request (UpReq): one HMAC verification to verify the authenticity of the message and one RSA decryption to obtain the foreign index pseudonym. Upon the generation of the uploading response (UpRes): it performs one HMAC signature generation to protect the authenticity of the response message.

7.7.1.2 Foreign2HomeServer Protocol

As described in section 7.4.2, the cryptographic operations during the execution of the Foreign2HomeServer protocol are as follows.

- Upon the generation of a forwarding request (FwReq), the data collection server (sender) performs one HMAC signature to protect the authenticity of the message. Upon receiving a forwarding response (FwRes) the data collection server performs one HMAC signature verification to verify the authenticity of response message.
- The data collection server (receiver) performs the following operations upon receiving the forwarding request (FwReq): one HMAC verification to verify the authenticity of the message, (m) RSA decryptions to obtain the home index pseudonym, (n) AES decryptions to obtain the second MAC and EMPGHD for each patient, and (n) HMAC verifications to verify the authenticity of the second MAC of each EMPGHD. Upon the generation of the forwarding response (FwRes), the receiver generates one HMAC signature to protect the authenticity of the response message.

7.7.1.3 Home2HCPServer Protocol

As described in section 7.4.3, the cryptographic operations during the execution of the protocol are as follows.

- The data collection server (sender) performs one HMAC generation upon the generation of a forwarding request (HfwReq). Upon receiving a forwarding response (HfwRes) it performs one HMAC signature verification to verify the authenticity of response message.
- The healthcare provider (HCP) performs the following operations upon receiving the forwarding request (HfwReq): one HMAC verification to verify the authenticity of the message, (m) AES decryptions to get the patient real ID, (n) AES decryptions to get the first MAC and PGHD for each patient, and (n) HMAC verifications to verify the authenticity of the first MAC of each PGHD. Upon the generation of a forwarding response (HfwRes), the HCP generates one HMAC signature generation to protect the authenticity of the response message.

7.7.2 Communication Cost

Here we find the number of bytes carried in each message of each protocol.

7.7.2.1 The Data Uploading Protocol

In the data uploading protocol, as described in section 7.4.1, the patient uploads authenticated data to a foreign server. This protocol is executed many times until the patient finishes all uploads. The communication overhead for the protocol is calculated as follows.

- The patient sends this request message,
  $\mathrm{UpReq} = (\mathrm{FRP}^{\mathrm{f}}_{\mathrm{i,r}}||\mathrm{ID_f}||\mathrm{TagNo}||\mathrm{2EMPGHD}||\mathrm{HMAC})$
- The foreign server responds with,
  $\mathrm{UpRes} = (\mathrm{ID_f}||\mathrm{FRP}^{\mathrm{f}}_{\mathrm{i,r}}||\mathrm{ACK}||\mathrm{HMAC})$.

Thus, the communication overhead introduced by the protocol when it runs from the patient to the foreign server is as follows. The $\mathrm{FRP}^{\mathrm{f}}_{i,r}$ size is 272 bytes, assuming the 2EMPGHD size is around 850 bytes based on Figure 7.4, the $\mathrm{ID_f}$ size is 4 bytes, the TagNo size is 4 bytes, and the HMAC size is 64 bytes. The total communication overhead introduced by the UpReq is 1194 bytes, i.e., 272+4+4+850+64. The communication overhead introduced by the protocol when it runs from the foreign server to the patient is as follows. The $\mathrm{ID_f}$ size is 4 bytes, the $\mathrm{FRP}^{\mathrm{f}}_{i,r}$ size is 272 bytes, the ACK size is 4 bytes, and the HMAC size is 64 bytes. The total communication overhead introduced by the UpRes is 344 bytes, i.e., 4+272+4+64.
The communication overhead for the protocol when using non-hierarchical pseudonym is calculated as follows.

- The patient sends a request message,
  $\mathrm{UpReq} = (\mathrm{nFRP}^{\mathrm{f}}_{\mathrm{i,r}}||\mathrm{ID_f}||\mathrm{2EMPGHD}||\mathrm{HMAC})$
- The foreign server responds with,
  $\mathrm{UpRes} = (\mathrm{ID_f}||\mathrm{nFRP}^{\mathrm{f}}_{\mathrm{i,r}}||\mathrm{ACK}||\mathrm{HMAC})$.

Thus, the communication overhead introduced by the protocol when it runs from the patient to the foreign server is as follows. The $\mathrm{nFRP}^{\mathrm{f}}_{i,r}$ size is 256 bytes, the 2EMPGHD size is 850 bytes, the $\mathrm{ID_f}$ size is 4 bytes, and the HMAC size is 64 bytes. The total communication overheads introduced by the UpReq is 1174 bytes, i.e., 256+4+850+64.

The communication overhead introduced by the protocol when it runs from the foreign server to the patient is as follows. The $\mathrm{ID_f}$ size is 4 bytes, the $\mathrm{FRP}^{\mathrm{f}}_{i,r}$ size is 256 bytes, the ACK size is 4 bytes, and the HMAC size is 64 bytes. The total communication overheads introduced by the UpRes is 328 bytes, i.e., 4+256+4+64

7.7.2.2 The Foreign2HomeServer Protocol

In the Foreign2HomeServer forwarding protocol, as described in section 7.4.2, each data collection server which worked as a foreign sever for some patients forwards patients' data to their home servers. Assume we have two data collection servers, one is the foreign server ($\mathrm{ID_s}$) and the other is the home server ($\mathrm{ID_r}$).

- The foreign server (ID_s) sends this request message,
  $FwReq = (ID_s||ID_r||AGG||HMAC)$.
- The home server (ID_r) responds with,
  $FwRes = (ID_r||ID_s||ACK||HMAC)$.

Thus, the communication overhead introduced by the protocol when it runs from the foreign server to the home server is as follows. The ID_s size is 4 bytes, the ID_r size is 4 bytes, the AGG size is up to 1 MB, and the HMAC size is 64 bytes. The total communication overhead introduced by the FwReq is 1000072 bytes, i.e., 4+4+1000000+64. The communication overhead introduced by the protocol when it runs from the home server to the foreign server is as follows. The ID_r size is 4 bytes, the ID_s size is 4 bytes, the ACK size is 4 bytes, and the HMAC size is 64 bytes. The total communication overheads introduced by the UpRes is 76 bytes, i.e., 4+4+4+64.

7.7.2.3 The Home2HCPServer Protocol

In the Home2HCPServer forwarding protocol, as described in section 7.4.3, each data collection server forwards patients' data to the HCP.

- The data collection server sends this request message,
  $HfwReq = (ID_s||ID_{HCP}||AGG||HMAC)$
- The HCP responds with,
  $HfwRes = (ID_{HCP}||ID_s||ACK||HMAC)$.

Thus, the communication overhead introduced by the protocol when it runs from the data collection server to the HCP is as follows. The ID_s size is 4 bytes, the ID_{HCP} size is 4 bytes, the AGG size is up to 1 MB, and the HMAC size is 64 bytes. The total communication overheads introduced by the HfwReq is 1000072 bytes, i.e., 4+4+1000000+64. The communication overhead introduced by the protocol when it runs from the HCP to the data collection server is as follows. The ID_{HCP} size is 4 bytes, the ID_s size is 4 bytes, the ACK size is 4 bytes, and the HMAC size is 64 bytes. The total communication overheads introduced by the HfwRes is 76 bytes, i.e., 4+4+4+64.

7.8 Performance Evaluation

The software which is used to implement the data uploading protocol is Java 2 Platform, Standard Edition (J2SE). Java is chosen because it supports a set of standard security interfaces. That is, it includes the Java Cryptography Extension (JCE). The JCE provides the implementations of several cryptographic primitives and key management services required in the uploading protocol, including a secure random number generator, key management facilities, a message digest function (SHA256), block ciphers (AES and RSA), elliptical curve

Table 7.2: Uploading using hierarchical pseudonym

Operations	Key Size	Time (ms)
The patient executes the following algorithms		
Retrieve DataBase	-	0.293
2 AesEncryption	256	0.004
1 AesEncryption	256	0.0
2 HmacGen256	256	0.016
1 HmacGen256	256	0.003
The foreign server executes the following algorithms		
FVK	256	21
AesDecryption	256	0.0
Store DataBase	256	3.4
HmacGen256	256	0.003
The patient executes the following algorithms		
HmacVer256	256	0.003

Table 7.3: Uploading using non-hierarchical pseudonym

Operations	Key Size	Time (ms)
The patient executes the following algorithms		
Retrieve DataBase	-	0.293
2 AesEncryption	256	0.004
RsaEncryption	2048	3.1
2 HmacGen256	256	0.016
HmacGen265	2048	0.003
The foreign server executes the following algorithms		
RsaDecryption	2048	3.1
EMul	256	0.1
HmacVer256	256	0.003
Store DataBase	256	3.4
HmacGen256	256	0.003
The patient executes the following algorithms		
HmacVer256	256	0.003

cryptosystem (ECC), blind signature, and digital certificate. The databases which are used in the protocol are created using MySQL [64]. In this section, we describe the design of the experiment and the evaluation metrics.

- The performance metrics used for the protocol evaluation are: The average time a patient spends waiting in a queue and the average response time of the system.

- Java Microbenchmark Harness (JMH): To measure the execution time for each protocol, we use a Java benchmarking tool called Java Microbenchmark Harness (JMH) [65]. Table 7.2 shows the time in ms for each method used in the protocol. The table shows the number of cryptographic methods used by each entity that participates in the protocol, the key size used for each method, and the time in ms required to execute each method. Each value in this table represents the mean value of 20 forks. For each fork, we ran 20 warm-up iterations. The value of warm-up iterations is ignored. After the 20 warm-up iterations, we ran 20 benchmark iterations. The same is true for Table 7.3 which shows the time required to execute each method used in the protocol.

- Hardware Architecture: : To prototype the protocol, a desktop computer running Windows 10 with a 1.99 GHz Intel Core i7 and 8GB of RAM was used. The timing results from the execution of the protocol presented here are based on this computer's specifications.

- Queuing theory [66] will be used to predicate the performance of the protocol.

7.8.1 Theoretical Analysis

In this section we measure the performance of the protocol using two models: the single and multiple servers queuing models. The mathematical formulas for the single queue model (M/M/1) and multiple server queuing models (M/M/C) were described in Chapter 6.

To theoretically analyze the performance of a protocol in terms of the waiting time and the total response time of the system, we first need to determine the service time (m), the arrival rates of the requests (λ), and the service rate (μ). The service time (m) is the time the server needs to conduct the calculation for each request. In other words, it is the sum of the execution time of each method executed at the server side during the verification of the request and creation of the response. For example, the service time for the protocol using hierarchical pseudonym is 0.024 second which is the sum of the execution time of the methods executed on the server listed in Table 7.2.

The service rate μ can be calculated as (1/m). The arrival rates (λ) in a single queuing model (M/M/1) should be less than μ, otherwise we will obtain negative values. In the multi-server queuing model (M/M/C), the arrival rates should be less than $\mu * s$, where s is the number of servers.

Table 7.4: Uploading using hierarchical pseudonym

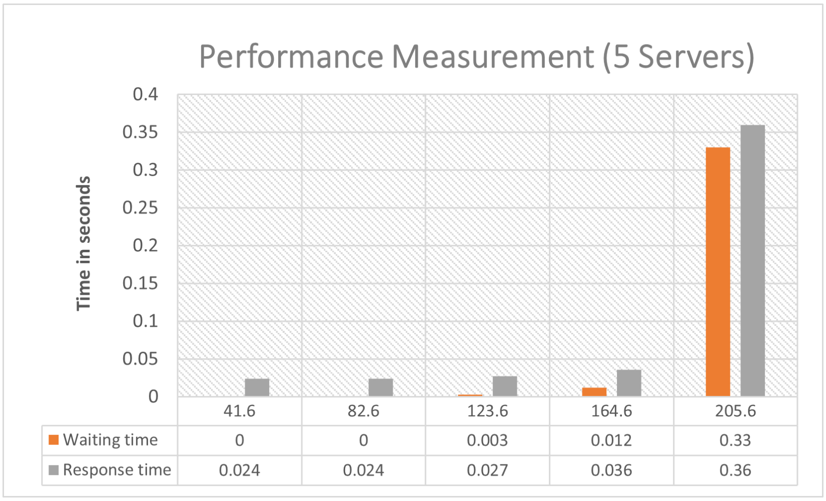
λ	ρ	L_Q	L	W_Q	W	P_0
Servers = 1						
20.5	49.2	0.48	0.97	0.023	0.047	50.8
25.5	61.2	0.96	1.57	0.038	0.062	38.85
30.5	73.14	1.99	2.72	0.065	0.089	25.859
35.5	85.13	4.87	5.73	0.137	0.161	14.86
40.5	97.1	32.77	33.750	0.809	0.833	2.87
Servers = 3						
25.5	19.98	0.006	0.606	0.0	0.024	54.8
50.5	40.36	0.098	1.31	0.002	0.03	29.1
75.5	60.35	0.547	2.4	0.007	0.031	14.4
100.5	80.33	2.6	5.1	0.027	0.051	5.5
125	99.9	1248	1251	9.985	10.01	0.02
Servers = 5						
41.67	19.98	0.001	1.0	0	0.024	36.80
82.67	39.65	0.038	2.020	0	0.024	13.6
123.6	59.28	0.331	3.295	0.003	0.027	4.856
164.6	78.9	2.002	5.949	0.012	0.036	1.408
205.6	98.61	68.44	73.37	0.333	0.357	0.055

7.8.1.1 Data Uploading Performance

In this section, we describe the performance of the protocol using hierarchical pseudonyms and non-hierarchical pseudonyms. To measure the performance of the protocol, we use queue

Table 7.5: Uploading using non-hierarchical pseudonym

λ	ρ	L_Q	L	W_Q	W	P_0
Servers = 1						
28.6	20.0	0.1	0.3	0.002	0.01	80
56.6	39.6	0.3	0.7	0.005	0.01	60.4
84.6	59.2	0.9	1.5	0.01	0.02	40.8
112.6	78.8	2.9	3.7	0.03	0.03	21.2
140.6	98.4	59.6	60.6	0.42	0.4	1.6
Servers = 3						
85.7	20	0.01	0.61	0	0.01	54.8
171.4	39.9	0.1	1.3	0	0.01	29.4
256.4	59.8	0.52	2.3	0	0.01	14.7
341.4	79.6	2.51	4.9	0	0.01	5.7
426.4	99.5	183.5	186.5	0.430	0.44	0.12
Servers = 5						
142.9	20	0	1	0	0.01	36.8
285.8	40	0.04	2	0	0.01	13.4
428.7	60	0.4	3.4	0	0.01	4.7
571.6	80	2.2	6.2	0	0.01	1.3
700.5	98	47.6	52.5	0.1	0.1	0.1

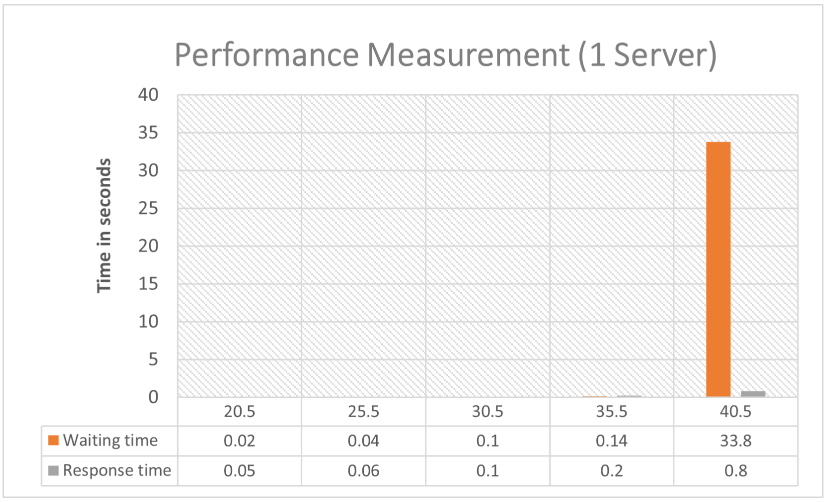


	41.6	82.6	123.6	164.6	205.6
■ Waiting time	0	0	0.003	0.012	0.33
■ Response time	0.024	0.024	0.027	0.036	0.36

Figure 7.5: Hierarchical uploading protocol performance using 5 servers.

modeling. In the single queuing model there is only one server, and the arrivals of patients' requests occurs at lambda (λ) of an exponential distribution. The service rate occurs at mu (μ) of the Poisson process. In the multi-server model, we select a different number of servers (3,5 servers).

- Single queuing model: As one can see from Table 7.4, we have calculated the performance of our system using the mathematical formulas for the single queue model (M/M/1). With one server the arrival rates should not exceed μ which is (1/0.024) = 41.7. We selected the range of arrival rates to be 20.5 to 40.5. After running the calculations using queuing formulas (presented in Chapter 6) we found the following: the minimum waiting time and response time occur when the arrival rate is 1 request per second. The maximum waiting time and response time occur when the arrival rate reaches 1.29 requests per second. The overall average of the waiting time and response time of our system using one server are 6.8 seconds and 0.24 seconds respectively. Figure 7.6 shows the performance measurement.



	20.5	25.5	30.5	35.5	40.5
🟧 Waiting time	0.02	0.04	0.1	0.14	33.8
⬜ Response time	0.05	0.06	0.1	0.2	0.8

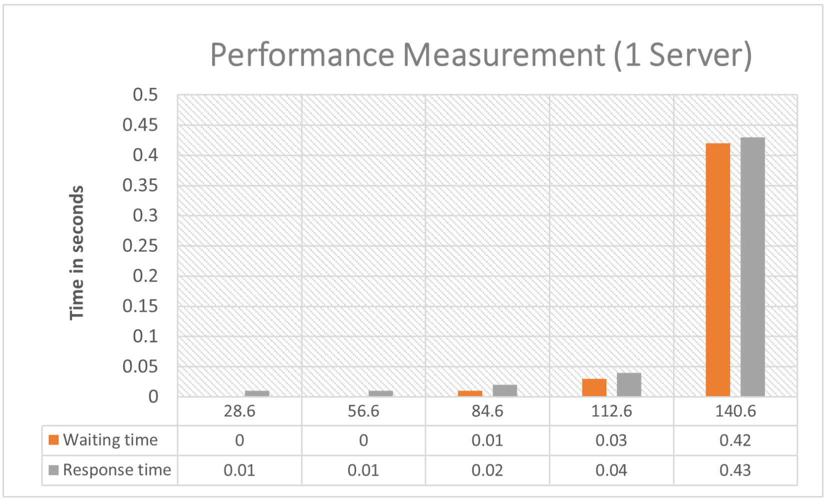
Figure 7.6: Hierarchical uploading protocol performance using 1 server.

- Multi-server queuing model (server number=3): As one can see from Table 7.4, we have calculated the performance of our system using the mathematical formulas for the multi-server queuing model (M/M/C). We have selected 3, 5 servers. In the case of 3 servers, the arrival rates should not exceed $\mu * s$ which is (41.7*3) = 125.1 requests per second. We selected the range of arrival rates to be 25.5 to 125. The minimum waiting time and response time occur when the arrival rate is 25.5 requests per second. The maximum waiting time and response time occur when the arrival rate reaches 125 requests per second. The overall average waiting time and response time of our system using 3 servers are 0.17 seconds and 0.19 seconds respectively. As one can see, we reduce the response time by 20.8% by using 3 servers.

- Multi-server queuing model (server number=5): In case of 5 servers, the arrival rates should not exceed $\mu * s$ which is (41.7*5) = 208.5 requests per second. We selected arrival rates ranging from 41.5 to 205.6. The minimum waiting time and response time occur when the arrival rate is 41.5 requests per second. The maximum waiting time and response time occur when the arrival rate reaches 205.6 requests per second. The overall average waiting time and response time for our system using 5 servers is 0.1 seconds. As one can see, we reduce the response time by 58.3% by increasing the number of servers. Figure 7.5 shows the performance measurement.

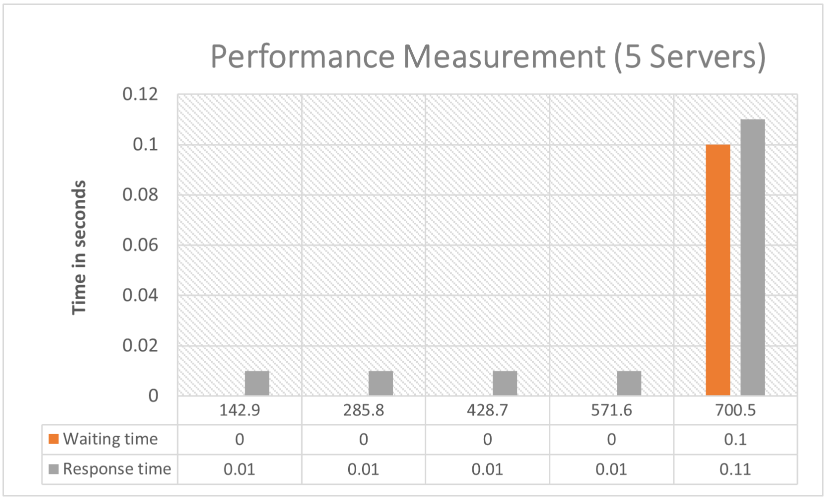
The performance of the protocol using a non-hierarchical pseudonym can be described as follows.

- Single queuing model: As one can see from Table 7.5, we have calculated the performance of our system using the mathematical formulas for the single queuing model (M/M/1). In this case, the arrival rate should not exceed μ which is (1/0.007) = 142.85. We selected the range of arrival rate to be 28.6 to 140.6. After running the calculations using the formulas we found the minimum waiting time and response time occur when the arrival rate is 28.6 requests per second. The maximum waiting time and response time occur when the arrival rate reaches 140.6 requests per second. The overall average waiting time and response time in



	28.6	56.6	84.6	112.6	140.6
🟧 Waiting time	0	0	0.01	0.03	0.42
⬜ Response time	0.01	0.01	0.02	0.04	0.43

Figure 7.7: Non-hierarchical uploading protocol performance using 1 server.



	142.9	285.8	428.7	571.6	700.5
🟧 Waiting time	0	0	0	0	0.1
⬜ Response time	0.01	0.01	0.01	0.01	0.11

Figure 7.8: Non-hierarchical uploading protocol performance using 5 servers.

our system using one server are 0.1 seconds and 0.1 seconds respectively. Figure 7.7 shows the performance measurement of the uploading protocol using one server.

- Multi-server queuing model (server number=3): As one can see from Table 7.4, we have calculated the performance of our system using the mathematical formulas for the multi-server queuing model (M/M/C). We have selected 3 and 5 servers. In the case of 3 servers, the arrival rates should not exceed $\mu * s$ which is (142.85*3) = 428.6 requests per second. We selected arrival rates ranging from 85.7 to 426.4. We can see that the waiting time and the response time are negligible.

- Multi-server queuing model (server number=5): In case of 5 servers, the arrival rates should not exceed $\mu * s$ which is (142.85*5) = 714.25 requests per second. We selected arrival rates ranging from 142.9 to 700.5. We can see that the waiting time and the response time are negligible. Figure 7.8 shows the performance measurement of the uploading protocol using 5 servers.

7.9 Comparison Studies

The A2DUF protocol suite can be considered as the first attempt to create a fully secured and ID privacy-preserving data uploading and forwarding solution in a distributed environment. Our solution allows a patient to select and upload the health data on any foreign servers after a successful authentication using digital certificates generated by the patient. The A2DUF breaks the link between the patient and his/her uploads. However, each foreign server can link the patient's data to the patient account known to the foreign server then deliver the patient's data to the patient's home server. The A2DUF solution satisfies a number of design requirements which have not been considered in any related works. It satisfies the requirement for mutual anonymous authentication (R1) by using the Anonymous Distributed Authentication (ADA) solution explained in Chapter 6. It satisfies the requirement for data confidentiality and authenticity (R2 and R3) by using the double signing and encryption method. It satisfies the requirement for uploading anonymity and unlinkability (R4) by using unlinkable pseudonyms. It satisfies the requirement for unpredictable uploading pattern (R5) by using a swarming strategy to hide the uploading pattern. It satisfies the requirement for the load distribution (R6) by enabling the patient's home server to link patient's data which is scattered across multiple foreign servers. Then, the home server is responsible for aggregating the patient's data and delivering it to the HCP. It satisfies on-event data uploading (R7) by encapsulating the patient's health status in a request pseudonym.

As far as we are concerned, the related works [67], [68] which are designed to collect data from patients in distributed environments have not considered a number of design requirements that A2DUF solution has considered. Table 7.6 summarizes a comparison of these, where F means false or does not satisfy the requirement, T means true or does satisfy the requirement, and P means it partially satisfies the requirement. In [68], the authors proposed a novel method to collect data from vehicles and store the data on the blockchain. Each vehicle uploads the data to any road-side unit (RSU), then all the data is sent to a server to be verified. After that it can be published on the blockchain. There are several drawbacks in this work. First, it uses one server to verify all the data collected by each RSU. This may cause a single point of failure in the system. Second, it does not considered anonymity in uploading, which means that any entity could easily link the upload to the vehicle and find the pattern of communication. In [67], the authors proposed a decentralized privacy-preserving healthcare blockchain for IoT. In this work, the patient uploads confidential and authenticated data on a cloud server. The cloud server verifies the data, generates a data block, generates a hash for the data block, and sends the hash of the data block to the blockchain. For anonymous transactions, the authors proposed a ring signature scheme. Each block has information about the sender (the patient) and the receiver (doctor). Thus, the blockchain network can link the patients to their doctors. In both works, each time the user is uploading to same server; The user is not allowed to select and change the servers. Moreover, the patient is not in charge of generating the pseudonyms and the certificates that are used to communicate with the servers.

Table 7.6: A2DUF comparison with related works.

Requirements	[68]	[67]	A2DUF
R1	F	T	T
R2	T	T	T
R3	T	T	T
R4	F	P	T
R5	F	F	T
R6	F	F	T
R7	F	F	T

7.10 Chapter Summary

This chapter has presented a novel Anonymous Authenticated Data Uploading and Forwarding (A2DUF) protocol suite. The A2DUF consists of methods and uploading and forwarding protocols. The purpose of the uploading protocol is to allow a patient to upload authenticated data to multiple foreign servers selected by the patient and at the same time hide the upload pattern of the patient using the swarming method. The purpose of the forwarding protocol is to allow an anonymous linking of the patient's data, which is scattered across multiple foreign servers by the home server. Then the patient's data is linked to his/her real ID by the HCP. The A2DUF solution satisfies a number of design requirements which have not been considered in any related works. The most important of these requirements is hiding the uploading pattern using a swarming strategy. The A2DUF uses expensive cryptographic operations only when needed. Otherwise, it uses inexpensive ones.

Chapter 8

Conclusion and Future work

In this thesis, we designed a solution to provide secure, ID privacy-preserving, and efficient data collection from mobile patients. This thesis addresses the problem of identifying the user based on his/her communication pattern. The solution involves using a number of ideas. This chapter summarises these ideas and recommends areas for future work.

8.1 Thesis Contributions

The work has led to the following novel contributions.

1. Design the SPDC framework: The framework uses many data collection servers owned by different service providers to collect the patient's data. This is to hide the pattern of interaction of the patient from any entity and to offer more scalability to the system. The SPDC architecture consists of a number of data collection servers owned by different service providers, and the healthcare provider's server. The patient selects one data collection server to be the home server, and a number of data collection servers to be the patient's foreign servers. Each foreign server collects data from the patient. The home server is responsible for collecting the patient's data which is scattered across different foreign servers and forwarding it to the HCP. The HCP is the ultimate storage for patient's data where it is processed and analyzed to make the decisions.

2. Design of a verification method to verify a patient's request pseudonym without searching for the verification key in a database: The non-hierarchical verification (nHPVer) method allows the server to verify the request pseudonym without searching for the verification key in the database. This is achieved by the way we designed the non-hierarchical pseudonym as an elliptical curve public key. So, when the non-hierarchical request pseudonym is received by the server, the server multiplies the request pseudonym with its elliptical curve private key. This process results in the verification key that is used to verify the MAC associated with the request pseudonym.

3. The patient generates digital certificates to access any data collection server: The patient generates the Foreign Pseudonym Certificates (FCerts) in cooperation with the home server to register with any data collection server as a foreign server. The patient generates the certificates and the home server signs them blindly.

4. Design of swarming method: The patient uses the swarming algorithm to upload the data to any foreign servers and to prevent any entity from learning the pattern of uploads.

5. Design double signing and encryption method: This method ensures that no entity can generate data on behalf of the patient and no entity can learn anything about the patient's data except the HCP.

6. The SPDC framework has the following features that are not found in other frameworks: The SPDC allows the patient to select where to cache the data and who is responsible for delivering the data to the final destination (HCP). The SPDC allows the patient to generate certificates and pseudonym IDs to access any servers. In SPDC, the authentication per session can be done without searching in the verifier's database for the verification key. The SPDC allows the patient to double sign and encrypt the data before upload so no data collection servers know the content of the patient's data. The SPDC allows the patient to not be recognized based on the communication pattern by using: (1) multiple data collection servers and they are changeable every day, and (2) the swarming algorithm which allows the patient to upload randomly on the foreign servers.

7. The SPDC framework uses a combination of different cryptographic building blocks to enhance performance: In chapter 5, the hierarchical request pseudonyms are generated using AES encryption. This means that to verify the request pseudonym, the server needs to find the key used to decrypt the pseudonym and then use the same key to verify the MAC attached with the message. To make the verification fast without searching in the database for the verification key, we used RSA encryption. The non-hierarchical request pseudonym is generated based on an elliptical curve public key (EPK) and some random text. Then, the EPK is encrypted using the server's RSA public key to form the non-hierarchical request pseudonym. So when the pseudonym arrives at the server, the server decrypts the pseudonym using the server's RSA private key to discover the EPK. Then, the server multiples the EPK with its elliptical curve private key (EPR) to get the verification key and verify the message. In chapter 6, we see that we used ECC based certificate instead of an RSA certificate as the former has a smaller key size.

8. The SPDC framework uses multiple servers to serve many patients so the processing time may be insignificant at a server side. However, the processing time on the patient's mobile device is significant. The processing time when the patient is using the hierarchical authentication protocol is insignificant on the mobile device but the amount of time it takes for a server to respond to the patient's request is large as a result of searching in the database to find the verification key. This is not the case when the patient is using the non-hierarchical authentication protocol, the processing time on the mobile device is noticeable but the response time is small as the server does not search in the database. The communication overheard of an uploading message depends on the type of the sensor that generates the data. For example, the communication overhead of a message that contains a video is different than the one that has text data (e.g., temperature data).

8.2 Future Work

The following provides recommendations for future work:

- The design of the SPDC protocols has considered only the security and privacy of one-way communication. For example, in the uploading protocol, when the foreign server sends an acknowledgment response to the patient upon receiving a request, the server uses the same request pseudonym that the patient used in the request message. For a high level of ID privacy, the server should generate a request pseudonym for the response message.

- The performance evaluation of SPDC has been done using a single machine. We use a desktop computer to represent the home server, foreign server, healthcare provider server (HCP), and the mobile device of the patient. For a correct measure of performance, we should have used four devices to represent each entity. This would help us to know the exact impact of running SPDC protocols on a mobile device, how long it takes to conduct authentication first, and then upload the data from the mobile device to the server.

- In this thesis, we only write the procedure of the swarming algorithm without a real implementation. For future work, we will implement this algorithm and determine the impact of the algorithm on the performance of the uploading protocol.

- For the forwarding protocols, we calculated the cost and communication overhead. The protocol implementation is left for future work.

References

[1] M. reporter, “The nhs is about to take an ‘important’ step into the cloud, says microsoft,” Jan. 2018. [Online]. Available: `https://news.microsoft.com/en-gb/2018/01/19/the-nhs-is-about-to-take-an-important-step-into-the-cloud-says-microsoft/`.

[2] P. Kakria, N. Tripathi, and P. Kitipawang, “A real-time health monitoring system for remote cardiac patients using smartphone and wearable sensors,” *International journal of telemedicine and applications*, vol. 2015, p. 8, 2015.

[3] J. Lopez and R. Dahab, “An overview of elliptic curve cryptography,” 2000.

[4] T. T. Tracy, “The wonder of birds: What they tell us about ourselves, the world, and a better future,” *Perspectives on Science and Christian Faith*, vol. 71, no. 2, pp. 124–126, 2019.

[5] K. T. Kadhim, A. M. Alsahlany, S. M. Wadi, and H. T. Kadhum, “An overview of patient’s health status monitoring system based on internet of things (iot),” *Wireless Personal Communications*, pp. 1–28, 2020.

[6] S. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K.-S. Kwak, “The internet of things for health care: A comprehensive survey,” *IEEE Access*, vol. 3, pp. 678–708, 2015.

[7] G. Paliwal and A. W. Kiwelekar, “A comparison of mobile patient monitoring systems,” in *International Conference on Health Information Science*, Springer, 2013, pp. 198–209.

[8] P. Pawar, V. Jones, B.-J. F. Van Beijnum, and H. Hermens, “A framework for the comparison of mobile patient monitoring systems,” *Journal of biomedical informatics*, vol. 45, no. 3, pp. 544–556, 2012.

[9] D. J. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton, “Codeblue: An ad hoc sensor network infrastructure for emergency medical care,” in *International workshop on wearable and implantable body sensor networks*, 2004.

[10] A. Wood, G. Virone, T. Doan, Q. Cao, L. Selavo, Y. Wu, L. Fang, Z. He, S. Lin, and J. Stankovic, “Alarm-net: Wireless sensor networks for assisted-living and residential monitoring,” *University of Virginia Computer Science Department Technical Report*, vol. 2, p. 17, 2006.

[11] J. Ko, J. H. Lim, Y. Chen, R. Musvaloiu-E, A. Terzis, G. M. Masson, T. Gao, W. Destler, L. Selavo, and R. P. Dutton, "Medisn: Medical emergency detection in sensor networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 10, no. 1, pp. 1–29, 2010.

[12] F. Hu, M. Jiang, L. Celentano, and Y. Xiao, "Robust medical ad hoc sensor networks (masn) with wavelet-based ecg data mining," *Ad Hoc Networks*, vol. 6, no. 7, pp. 986–1012, 2008.

[13] A. Van Halteren, R. Bults, K. Wac, D. Konstantas, I. Widya, N. Dokovsky, G. Koprinkov, V. Jones, and R. Herzog, "Mobile patient monitoring: The mobihealth system," *Journal on Information Technology in Healthcare*, vol. 2, no. 5, pp. 365–373, 2004.

[14] M. N. K. Boulos, A. Rocha, A. Martins, M. E. Vicente, A. Bolz, R. Feld, I. Tchoudovski, M. Braecklein, J. Nelson, G. Ó. Laighin, *et al.*, *Caalyx: A new generation of location-based services in healthcare*, 2007.

[15] P. Gope and T. Hwang, "Bsn-care: A secure iot-based modern healthcare system using body sensor network," *IEEE sensors journal*, vol. 16, no. 5, pp. 1368–1376, 2015.

[16] G. Almashaqbeh, T. Hayajneh, A. V. Vasilakos, and B. J. Mohd, "Qos-aware health monitoring system using cloud-based wbans," *Journal of medical systems*, vol. 38, no. 10, p. 121, 2014.

[17] J. H. Abawajy and M. M. Hassan, "Federated internet of things and cloud computing pervasive patient health monitoring system," *IEEE Communications Magazine*, vol. 55, no. 1, pp. 48–53, 2017.

[18] M. Quwaider and Y. Jararweh, "Cloudlet-based efficient data collection in wireless body area networks," *Simulation Modelling Practice and Theory*, vol. 50, pp. 57–71, 2015.

[19] S. R. Moosavi, T. N. Gia, E. Nigussie, A. M. Rahmani, S. Virtanen, H. Tenhunen, and J. Isoaho, "End-to-end security scheme for mobility enabled healthcare internet of things," *Future Generation Computer Systems*, vol. 64, pp. 108–124, 2016.

[20] E. Marin, M. A. Mustafa, D. Singelée, and B. Preneel, "A privacy-preserving remote healthcare system offering end-to-end security," in *International Conference on Ad-Hoc Networks and Wireless*, Springer, 2016, pp. 237–250.

[21] W. Tang, K. Zhang, D. Zhang, J. Ren, Y. Zhang, and X. S. Shen, "Fog-enabled smart health: Toward cooperative and secure healthcare service provision," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 42–48, 2019.

[22] J. H. Abawajy and M. M. Hassan, “Federated internet of things and cloud computing pervasive patient health monitoring system,” *IEEE Communications Magazine*, vol. 55, no. 1, pp. 48–53, 2017.

[23] G. Wang, R. Lu, and Y. L. Guan, “Achieve privacy-preserving priority classification on patient health data in remote ehealthcare system,” *IEEE Access*, vol. 7, pp. 33 565–33 576, 2019.

[24] R. Boussada, B. Hamdane, M. E. Elhdhili, and L. A. Saidane, “Privacy-preserving aware data transmission for iot-based e-health,” *Computer Networks*, vol. 162, p. 106 866, 2019.

[25] R. Lu, X. Lin, and X. Shen, “Spoc: A secure and privacy-preserving opportunistic computing framework for mobile-healthcare emergency,” *IEEE transactions on parallel and distributed systems*, vol. 24, no. 3, pp. 614–624, 2012.

[26] X. Liang, R. Lu, L. Chen, X. Lin, and X. Shen, “Pec: A privacy-preserving emergency call scheme for mobile healthcare social networks,” *Journal of Communications and Networks*, vol. 13, no. 2, pp. 102–112, 2011.

[27] X. Lin, R. Lu, X. Shen, Y. Nemoto, and N. Kato, “Sage: A strong privacy-preserving scheme against global eavesdropping for ehealth systems,” *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 4, pp. 365–378, 2009.

[28] Q. Shen, X. Liang, X. S. Shen, X. Lin, and H. Y. Luo, “Exploiting geo-distributed clouds for a e-health monitoring system with minimum service delay and privacy preservation,” *IEEE journal of biomedical and health informatics*, vol. 18, no. 2, pp. 430–439, 2013.

[29] M. Quwaider and Y. Jararweh, “Cloudlet-based efficient data collection in wireless body area networks,” *Simulation Modelling Practice and Theory*, vol. 50, pp. 57–71, 2015.

[30] J. Xu, K. Xue, S. Li, H. Tian, J. Hong, P. Hong, and N. Yu, “Healthchain: A blockchain-based privacy preserving scheme for large-scale health data,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8770–8781, 2019. doi: 10.1109/JIOT.2019.2923525.

[31] M. H. Eiza, Q. Ni, and Q. Shi, “Secure and privacy-aware cloud-assisted video reporting service in 5g-enabled vehicular networks,” *IEEE Transactions on Vehicular Technology*, vol. 65, no. 10, pp. 7868–7881, 2016.

[32] J. Ni, X. Lin, K. Zhang, and X. Shen, “Privacy-preserving real-time navigation system using vehicular crowdsourcing,” in *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*, IEEE, 2016, pp. 1–5.

[33] C. Huang, H. Lee, H. Kim, and D. H. Lee, "Mvsers: A secure emergency response solution for mobile healthcare in vehicular environments," *The Computer Journal*, vol. 58, no. 10, pp. 2461–2475, 2015.

[34] J. Hoffstein, J. Pipher, J. H. Silverman, and J. H. Silverman, *An introduction to mathematical cryptography*. Springer, 2008, vol. 1.

[35] M. Mambo, K. Usuda, and E. Okamoto, "Proxy signatures: Delegation of the power to sign messages," *IEICE transactions on fundamentals of electronics, communications and computer sciences*, vol. 79, no. 9, pp. 1338–1354, 1996.

[36] S. Lal and A. K. Awasthi, "Proxy blind signature scheme," *Journal of Information Science and Engineering*, vol. 72, 2003.

[37] D. Chaum, "Blind signature system," in *Advances in cryptology*, Springer, 1984, pp. 153–153.

[38] C.-H. Wang and M.-Z. Liao, "Security analysis and enhanced construction on ecdlp-based proxy blind signature scheme," *International Journal of e-Education, e-Business, e-Management and e-Learning*, vol. 4, no. 1, p. 47, 2014.

[39] R. Housley, W. Polk, W. Ford, and D. Solo, "Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile," 2002.

[40] E. Birrell and F. B. Schneider, "Federated identity management systems: A privacy-based characterization," *IEEE security & privacy*, vol. 11, no. 5, pp. 36–48, 2013.

[41] N. Zhang, A. Rector, I. Buchan, Q. Shi, D. Kalra, J. Rogers, C. Goble, S. Walker, D. Ingram, and P. Singleton, "A linkable identity privacy algorithm for healthgrid," *Studies in health technology and informatics*, vol. 112, pp. 234–246, 2005.

[42] L. L. Iacono, "Multi-centric universal pseudonymisation for secondary use of the ehr.," *Studies in health technology and informatics*, vol. 126, p. 239, 2007.

[43] R. Addas and N. Zhang, "An enhanced approach to supporting controlled access to eprs with three levels of identity privacy preservations," in *Symposium of the Austrian HCI and Usability Engineering Group*, Springer, 2011, pp. 547–561.

[44] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[45] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf, "Pseudonym systems," in *International Workshop on Selected Areas in Cryptography*, Springer, 1999, pp. 184–199.

[46] J. Camenisch and A. Lehmann, "Privacy-preserving user-auditable pseudonym systems," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2017, pp. 269–284.

[47] C. E. Shannon, “A mathematical theory of communication,” *ACM SIGMOBILE mobile computing and communications review*, vol. 5, no. 1, pp. 3–55, 2001.

[48] M. Shu, D. Yuan, C. Zhang, Y. Wang, and C. Chen, “A mac protocol for medical monitoring applications of wireless body area networks,” *Sensors*, vol. 15, no. 6, pp. 12 906–12 931, 2015.

[49] D. Recordon and D. Reed, “Openid 2.0: A platform for user-centric identity management,” in *Proceedings of the second ACM workshop on Digital identity management*, ACM, 2006, pp. 11–16.

[50] S. Cantor and T. Scavo, “Shibboleth architecture,” *Protocols and Profiles*, vol. 10, p. 16, 2005.

[51] C. Paquin, “U-prove technology overview v1. 1,” *Microsoft Corporation Draft Revision*, vol. 1, 2011.

[52] A. Dey and S. Weis, “Pseudoid: Enhancing privacy in federated login,” 2010.

[53] J. Maheswaran, D. Jackowitz, E. Zhai, D. I. Wolinsky, and B. Ford, “Building privacy-preserving cryptographic credentials from federated online identities,” in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, 2016, pp. 3–13.

[54] M. Isaakidis, H. Halpin, and G. Danezis, “Unlimitid: Privacy-preserving federated identity management using algebraic macs,” in *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, 2016, pp. 139–142.

[55] R. Laborde, A. Oglaza, S. Wazan, F. Barrere, A. Benzekri, D. W. Chadwick, and R. Venant, “A user-centric identity management framework based on the w3c verifiable credentials and the fido universal authentication framework,” in *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, 2020, pp. 1–8.

[56] G. Bianchi, M. Bonola, V. Falletta, F. S. Proto, and S. Teofili, “The sparta pseudonym and authorization system,” *Science of Computer Programming*, vol. 74, no. 1-2, pp. 23–33, 2008.

[57] T. Lenz and V. Krnjic, “Towards domain-specific and privacy-preserving qualified eid in a user-centric identity model,” in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, IEEE, 2018, pp. 1157–1163.

[58] S. S. Chow, Y.-J. He, L. C. Hui, and S. M. Yiu, "Spice–simple privacy-preserving identity-management for cloud environment," in *International Conference on Applied Cryptography and Network Security*, Springer, 2012, pp. 526–543.

[59] C. Li, Q. Wu, H. Li, and J. Liu, "Trustroam: A novel blockchain-based cross-domain authentication scheme for wi-fi access," in *International Conference on Wireless Algorithms, Systems, and Applications*, Springer, 2019, pp. 149–161.

[60] Y. Yao, X. Chang, J. Mišić, V. B. Mišić, and L. Li, "Bla: Blockchain-assisted lightweight anonymous authentication for distributed vehicular fog services," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3775–3784, 2019.

[61] W. Wang, N. Hu, and X. Liu, "Blockcam: A blockchain-based cross-domain authentication model," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, IEEE, 2018, pp. 896–901.

[62] K. Kaur, S. Garg, G. Kaddoum, F. Gagnon, and S. H. Ahmed, "Blockchain-based lightweight authentication mechanism for vehicular fog infrastructure," in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, IEEE, 2019, pp. 1–6.

[63] Ofcom, "Uk home broadband performance: A consumer summary of fixed-line broadband performance provided to residential consumers," *ofcom.org.uk*, 2016.

[64] C. Bell, *Introducing the MySQL 8 document store*. Springer, 2018.

[65] *Code tools jmh*, 2019. [Online]. Available: `https://openjdk.java.net/projects/code-tools/jmh/`.

[66] C.-H. Ng and S. Boon-Hee, *Queueing modelling fundamentals: With applications in communication networks*. John Wiley & Sons, 2008.

[67] A. D. Dwivedi, G. Srivastava, S. Dhar, and R. Singh, "A decentralized privacy-preserving healthcare blockchain for iot," *Sensors*, vol. 19, no. 2, p. 326, 2019.

[68] Q. Kong, L. Su, and M. Ma, "Achieving privacy-preserving and verifiable data sharing in vehicular fog with blockchain," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

Appendices

Appendix A

ADA Prototyping and Performance Evaluation

In this section, we describe the design of the anonymous distribution authentication (ADA) protocols in terms of the programming language, the hardware platform, and the database used to prototype the protocols. The anonymous distribution authentication (ADA) protocols are the inter-domain registration and intra-domains authentication protocols. Then we evaluate the performance of the protocols using the queuing theory.

A.0.1 Design inter-domains registration Protocols

The inter-domain registration protocol has two forms: the home server registration (HSR) protocol, and the foreign server registration (FSR) protocol. The inter-domain registration protocols consist of three participants, a patient (Alice), a healthcare provider (HCP), and a number of data collection servers. Each participant has its own database. Each data collection server has two interfaces and two databases. The home interface interacts with the patient who wants to use the server as a home server. The foreign interface for the patient who wants to use the server as a foreign server. In addition, each server has a home database (H-Database) and a foreign database (F-Database). The role of each entity is as follows:

- A patient, Alice, performs home and foreign server registration requests. To do so, she needs to (1) select one of the data collection servers to be the home server, while the others are foreign servers, and (2) prepare her request to be sent to the selected home server.
- The data collection server receives the request from Alice then performs the necessary verifications, operations, and sends the response to Alice.
- The healthcare provider (HCP): All participants should first register with HCP and receive their identities and certificates from the HCP.

In the following, we describe the design of the home and foreign registration protocol in two phases, the set-up phase (offline) and the registration phase (online).

- A. The set-up phase describes the operations performed by the entities prior the registration, and is done once in the life-time of the protocol implementation.

  - Public Key and Private Key Generation. Each participant in the protocol generates its own public/private key pair. The public keys are certified by the certification authority (HCP). First, the requesting patient (Alice) generates an ECC public/private key pair to be used for generating the shared key and signing her request. Alice sends the ECC public key to the HCP which issues a certificate called home pseudonym certificate (HCert). The HCert is known to Alice, the HCP, and Alice's home server. Second, each data collection server generates its own ECC and RSA public/private key pairs. Then it sends both public keys to the HCP to be certified in two certificates, one for the RSA public key and one for the ECC public key. Each data collection uploads both certificates to an open-access database which can be accessed by all entities in the system.
  - Patient's Home Index Pseudonym Generation: The HCP generates the home index pseudonym of the patient using two symmetrical keys of 128 bit length and using the HIP-Gen algorithm.
  - Digital Certificate Generation. The HCP generates the home pseudonym certificate (HCert) using the BouncyCastle library which is a Java library that complements the default Java Cryptographic Extension (JCE). The HCP uses the ECC public key that is generated by the patient and the HIP that is generated by the HCP and binds them in an HCert. This certificate is sent to Alice.
  - Database Creation. (1) Alice creates her database, which stores necessary identifiers and certificates. (2) Each data collection server creates two types of databases, one is the home database (H-Database) and the other is the foreign database (F-DataBase). The H-Database is used for the home registered patient. The H-Database contains the patient's home index pseudonym, the home pseudonym certificate, and the patient's health data collected by Alice's home server from Alice's foreign servers (which will be described in Chapter 7). The F-DataBase contains the patient foreign index pseudonym, the foreign pseudonym certificate, and the patient's uploaded data.

- B. Home Registration Phase: This phase describes the operations performed for home registration by Alice and the home server.

  - Creating the request. This is done by the requesting patient (Alice), where she creates her request message (hRegReq) and sends it to the home server. This includes generating the signature, generating the home shared key (HSK$_i$), and encrypting the HIP and HCert. Then, Alice constructs the request message and finally sends the request message to the home server.
  - Home server verification. This is done by the home server which receives the hRegReq message from Alice. It first decrypts the body of the message to get the HIP and HCert. The home server verifies both the HCert and Alice's signature the message. After the verification, the home server stores the HCert and the HIP in the H-Database. Then, based on the ECC public key stated in the HCert, the home server generates the home shared key (HSK$_i$).

  - Creating the response. This is done by the home server and includes generating the tag range and index nonce. After that, the server generates MAC using the shared key, constructs the response message (hRegRes), and sends it to Alice.
  - Patient verification. This part is done by Alice, when she receives the response (hRegRes) from the server, she verifies MAC using the home shared key, decrypts the message, and stores the tag range and index nonce in its database.

- C. Foreign Registration Phase. This phase describes the operations performed for foreign registration by Alice and the foreign server.

  - Creating the request. This part is done by the requesting patient (Alice), who creates the request message (fRegReq) and sends it to the foreign server. This includes generating the signature, generating the foreign shared key (FSK_i^f), and encrypting the FIP and FCert. Then, Alice constructs the request message and finally sends the request message to the foreign server.
  - Foreign server verifications. This part is done by the foreign server. In this part, the foreign server receives the fRegReq message from Alice and performs the necessary operations. It first decrypts the body of the message to get the FIP and FCert. The foreign server verifies both the FCert and Alice's signature. After the verification, the foreign server stores the FCert and the FIP in the F-Database. Then, based on the ECC public key stated in the FCert, the foreign server generates the foreign shared key (FSK_i^f).
  - Creating the response. This part is done by the foreign server and includes generating the tag range and index nonce. After that, the server generates MAC using the shared key, constructs the response message (fRegRes) and sends it to Alice.
  - Patient verifications. This is done by Alice, when she receives the response (fRegRes) from the foreign server, she verifies MAC using the foreign shared key, decrypts the message, and stores the tag range and index nonce in its database.

A.0.2 The Implementation of Inter-Domains Registration Protocols

This section describes the classes and their methods which are designed for the HSR and FSR protocols. Some of these classes are executed prior to the runing of the protocols. Others are executed during the run of the protocols. The registration (HSR and FSR) protocols consist of five classes. Two of these classes are run-time classes: (1) Class Reg on (patient) and (2) Class PatReg on (data collection server). The other three are set-up classes, i.e., they are classes being executed prior to the protocol run: (3) Class CGen and (4) HCertGen on HCP. The CGen generates certificates for data collection servers and the HCertGen generates the home index pseudonym and ECC home pseudonym certificate (HCert) for patients. (5) Class Main, which contains general cryptographic methods, can be accessed by all classes. In the following, we explain these classes along with the methods used in them.

- Class Reg - This class runs on the patient's mobile device and it contains the following methods: CreateRegReq(), VerifyRes (), and doECDH().
- Class PatReg - This class runs on a data collection server and it contains the following methods: CreateRegRes(), VerifyReq(), doECDH(), and TagCalculate().
- Class CGen - This class runs on HCP and uses the CreateCert() method to generate a certificate for each data collection server.
- Class HCertGen - This class uses the following methods: the HIP-Gen() and PCreateCert() methods.
- Class Main - This class uses the following methods: AesEncryption(), AeskeyGen(), AesDecryption(), RsaEncryption(), RsaDecryption, HmacGen256(), HmacVer256(), Hash256(), EKeyGen(), ESigGen(), ESigVer(), getTime(), SecureRandom(), and CertVer().

  The operations performed during the set-up, home and foreign registration phases are implemented as follows.

Set-up phase operations' implementation:

- Generating the home index pseudonym (HIP). This is implemented by running the class HCertGen to generate a patient's HIP. Firstly, it generates the patient's main index pseudonym (MIP) using the HIP-Gen method. To do so, the HIP-Gen method uses the AeskeyGen() method (from the Main class) to generate two AES keys. The keys are used for all patients. The HCP uses the getTime()method (from the main class) to generate the pseudonym specific time. The HCP retrieves the patient real ID (PID) from the database. It then appends the time and the PID. The resultant text is then encrypted using AesEncryption() (from the main class) and the AES first key. The encryption process result in the MIP. Secondly, the HCP generates the HIP based on the MIP. To do so, the HCP uses the method SecureRandom (from the main class) to generate a random value. Then, it appends the MIP and the random value. The appended text is encrypted using AesEncryption() and the second AES key. This results in the HIP.
- Generating the certificates: This is implemented by running the class HCertGen to generate a home pseudonym certificate (HCert). To do this the HCP retrieves both the ECC public key of the patient (EPK) and the home index pseudonym from the database. It then sends the EPK and the HIP to PCreateCert() method to create the fields of the certificate and then signs the certificate with the HCP's ECC private key. Then, the certificate is sent to the patient and stored in the database.
- Creating the databases: The database for each entity is created using MySQL using the SQL query CREATE DATABASE. Then, for each database, tables are created using the method createTable.

Home registration phase operations' implementation:

- Creating the request: This is implemented by having the patient (Alice) run the Reg class. In this class, Alice creates her request using the method CreateRegReq(). In this method, the mobile device retrieves her home index pseudonym (HIP), home pseudonym certificate (HCert), and the RSA and ECC public keys of the home server from its database. It then appends the HIP and HCert. The appended string is then sent to the ESigGen() method in the Main class to generate a signature on the appended string. Then, the appended string is encrypted with the home server RSA public key using RsaEncryption() (from the main class). Then, the patient's mobile device sends the ECC public key of the home server and its ECC private key to the doECDH() method. This is to generate the home shared key (HSK). Then, the hRegReq request is sent to class PatReg on the home server.

- Server-side verification: Once the PatReg class receives the hRegReq from Alice, it decrypts the body of the message then starts the verification process. To do this, the home server calls the VerifyReq() method. In this method, the home server calls three methods from the main class: RsaDecryption(), CertVer(), and ESigVer(). The home server first decrypts the message body using its RSA private key. This is to obtain the HCert and HIP of the patient. Then, the home server verifies the HCert using the CertVer() method. This method verifies that the HCert is still valid (i.e., within 24 hours). Then, the ESigVer() method verifies the patient's signature on the message using the patient's ECC public Key stated in the HCert and verifies the signature of HCP on the HCert using HCP's ECC public key. If all verification is correct, the home server performs two operations. The first is to insert the HIP and HCert of the patient into the H-Database. Then, it calculates the home shared key using the doEHDC() method. Then, the home server inserts the home shared key into a table created for shared keys. Then, the home server calls method TagCalculate(), which takes the index where the home shared key is stored in the table and returns the min and max inputs. Subsequently, the home server initializes the index nonce which is a string 12 to 60 bits long.

- Creating the response: This achieved by calling the CreateRegRes() method. In this method the home server calls the following methods from the main class: the AesEncryption() and HmacGen256(). The AesEncryption() encrypts the tag range and the index nonce using the home shared key. Then, the HmacGen256() generates a MAC on the hRegRes message using the home shared key. The message is then sent to the Reg class on the patient's side.

- Patient-side verification: Once Alice receives hRegRes from the home server, shes uses the method VerifyReq() to verify the response from the server. The VerifyReq() method calls the following methods: HmacGen256() and AesDecryption(). HmacGen256() uses the home shared key to verify the MAC on the message. Then, the patient uses the home shared key as an input to the method AesDecryption() to decrypt the encrypted part and get the tag and the index nonce. Alice then stores the information in the database.

Foreign registration phase implementation of operations:

- Creating the request: This is implemented by having the patient Alice run the Reg class. In this class, Alice creates her request by using the method CreateRegReq(), in which the mobile device retrieves Alice's foreign index pseudonym (FIP), foreign pseudonym certificate (FCert), and the RSA and ECC public keys of the foreign server from its database. It then appends the FIP and FCert. The appended string is then sent to the ESigGen method in the main class to generate a signature on the appended string. Then, the appended string is encrypted with the foreign server RSA public key using RsaEncryption() method. Then, the patiet's mobile device sends the ECC public key of the foreign server and Alice's ECC private key to the doECDH() method. This is to generate the foreign shared key (FSK). Then the fRegReq request is sent to class PatReg on the foreign server.

- Server-side verification: Once the PatReg class receives fRegReq from Alice, it decrypts the body of the message then starts the verification process by using the VerifyReq() method. In this method, the foreign server calls four methods. These methods are RsaDecryption(), CertVer(), ESigVer(), and BlndSigVer(). The foreign server first decrypts the message body using its RSA private key. This is to obtain the FCert and FIP of the patient. Then, the foreign server verifies the FCert using the CertVer() method. This method verifies that the FCert is still valid (i.e., within 24 hours). Then, the ESigVer() method verifies the patient's signature on the message using the patient's ECC temporal public key stated in the FCert. The BlndSigVer() method verifies the signature of Alice's home server on the FCert using using Alice's home server proxy public key. If all verifications are correct, the foreign server performs two operations. The first is to insert the patient's FIP and FCert into the F-Database. It then calculates the foreign shared key using doEHDC(). Subsequently, the foreign server inserts the foreign shared key into a table created for shared keys. Then, the foreign server calls method TagCalculate() and initializes the index nonce.

- Creating the response: This achieved by calling the CreateRegRes() method. In this method the foreign server calls the following methods from the main class the AesEncryption() and HmacGen256(). The AesEncryption() encrypts the tag range and the index nonce using the foreign shared key. Then, the HmacGen256() generates a MAC on the fRegRes message using the foreign shared key. The message is then sent to Reg class on patient side.

- Patient-side verification: Once Alice receives fRegRes from the foreign server, she uses the method VerifyRes() to verify the response from the server. The VerifyRes() method calls the HmacGen256() and AesDecryption() methods. The HmacGen256() uses the home shared key to verify the MAC of the message. Then, the patient uses the foreign shared key as an input to the method AesDecryption() to decrypt the encrypted part and obtain the tag and the index nonce. Alice stores the information in the database.

A.0.3 The Design of Intra-Domain Authentication

The design of the intra-domain authentication protocol consists of two participants: a patient (Alice) and a data collection server. The authentication can be achieved by using a hierarchical or non-hierarchical request pseudonym. In this chapter the purpose of the authentication is to request a blind signature from the home server. In the next chapter, we will look at authentication for the purpose of uploading. The role of each entity is as follows:

- Alice performs the authentication request. To do so, she needs to (1) generate the foreign pseudonym certificate fields and hash them, and (2) prepare her blind signature request and send the request to the home server.
- The home server receives the request from Alice then performs verifications, operations, and sends the response (the blind signature) to Alice.

In the following, we describe the design of the intra-domains authentication using hierarchical and non-hierarchical request pseudonyms.

- A. Using a hierarchical request pseudonym involves the operations performed for home authentication by Alice and the home server.
  - Creating the request is done by the requesting patient (Alice), who creates her request message (BlndSigReq) and sends it to the home server. This includes generating a hierarchical home request pseudonym, foreign pseudonym certificate fields, and generating the MAC. Then, Alice constructs the request message and finally sends the request message to the home server.
  - Home server verifications involve the home server receiving the BlndSigReq message from Alice and performs the necessary operations. It first uses the tag number to find the segment that contains the verification key (home shared key). Then, the server iterates through the keys in the segment to find the key that verified the MAC. When the server finds the key, the key is used to decrypt the home request pseudonym and find the home index pseudonym and the index nonce. The index nonce is used to ensure the freshness of the request pseudonym. After verifying the patient, the home server creates the response.
  - Creating the response is done by the home server and includes generating the blind signature and the MAC using the home shared key. Then, it constructs the response message (BlndSigRes) and sends it to Alice.
  - Patient verification is done by Alice. When she receives the response (BlndSigRes) from the server, she verifies MAC using the home shared key, drives the signature from the blind signature, and attaches the signature to the foreign pseudonym certificate. The foreign pseudonym certificate is ready to be used.
- B. Using a non-hierarchical request pseudonym involves the operations performed for home authentication by Alice and the home server using non-hierarchical request pseudonym.
  - Creating the request is done by the requesting patient (Alice), who creates her request message (BlndSigReq) and sends it to the home server. This includes generating the

MAC, the non-hierarchical home request pseudonym, and generating the foreign pseudonym certificate fields. Then, Alice constructs the request message and finally sends the request message to the home server.

- Home server verifications is done by the home server. In this part, the home server receives the BlndSigReq message from Alice and performs the necessary operations. It first uses its RSA private key to decrypt non-hierarchical home request pseudonyms to obtain the elliptical curve public key of the patient and the index nonce. Then, the server multiplies its elliptical curve private key with the patient's elliptical curve public key to get the home shared key. The home shared key is used to verify the MAC on the message. After that, the server uses the index nonce to ensure the freshness of the request pseudonym. After the verification processes, the home server creates the response.
- Creating the response is done by the home server. This includes generating the blind signature and the MAC using the home shared key. Then the server constructs the response message (BlndSigRes) and sends it to Alice.
- Patient verification is done by Alice, when she receives the response (BlndSigRes) from the server, she verifies MAC using the home shared key, derives the signature from the blind signature, and attaches the signature to the foreign pseudonym certificate.

A.0.4 The Implementation of Intra-Domains Authentication Protocols

This section describes the classes and their methods which are used to implement for the FCertC protocol using hierarchical and non-hierarchical request pseudonyms. The protocol implementation consists of two classes (1) Class FCertCon on (patient) and (2) Class FCert-BlndSigGen on data collection server. In the following, we explain these classes along with the methods used in the classes.

- Class FCertCon: This class runs on the patient's mobile device and it contains the methods, CreateBlndSigReq(), VerifyRes (), HRPs-Gen(), nHRPs-Gen(), FIPs-Gen(), BlndSigDrv(), and BlndMsgGen().
- Class FCertBlndSigGen: This class runs on the data collection server and it contains the methods CreateBlndSigRes(), VerifyReq(), FVS(), HRPs-Lnk(), nHRPs-Lnk(), and Blnd-SigGen().

Authentication phase operations' implementation using hierarchical pseudonym:

- Creating the request is implemented by having the patient (Alice) run the FCertCon class. In this class, Alice creates her request using the method CreateBlndSigReq(). In this method, the mobile device generates a temporal elliptical curve key pair using the EKeyGen() method. Then, the mobile device generates the FCert data structure fields (i.e., the home ID, validity, and the subject). The subject is the foreign index pseudonym generated by the patient using the FIPs-Gen() method. Then, Alice generates the blind factors using BlndMsgGen () and hashes the FCert fields. Alice generates a hierarchical home request pseudonym using HRPs-Gen(). Then, the BlndSigReq message is constructed and MAC is generated using the HmacGen256() method. The message BlndSigReq is sent to class FCertBlndSigGen on the home server.

- Server-side verification: Once the home server receives BlndSigReq from Alice, the home server calls VerifyReq() method that uses the tag number as an input to the FVS() method. This method is used to find the home shared key and verify the message using the home shared key. The FVS() method uses the tag number to locate the segment. Then, it iterates through the segment and uses each key to verify the MAC using HmacGen256() from the main class. Then, the key which verified the MAC is used as an input to the HRPs-Lnk(). This is to find the home index pseudonym and the index nonce to ensure the freshness of the request pseudonym. The home index pseudonym is used to retrieve the HCert. The server uses CertVer() from the main class method to ensure that the HCert is within 24 hours and has not expired.

- Creating the response is achieved by calling the CreateBlndSigRes() method. In this method the home server calls the BlndSigGen() and HmacGen256() methods. The BlndSigGen() method generates the blind signature. Then, the home server prepares the BlndSigRes message and calls the HmacGen256() to generate the MAC on the message using the home shared key. The message is then sent to the patient.

- Patient-side verification occurs once Alice receives BlndSigRes from the home server. She then uses the method VerifyRes() to verify the response from the server. The VerifyRes() method calls the following methods, the HmacVer256() and BlndSigDrv(). The HmacVer256() uses the home shared key to verify the MAC of the message. Then, the patient uses the method BlndSigDrv() to unblind the blind signature. The signature is then attached to the FCert.

Authentication phase operations' implementation using the non-hierarchical pseudonym:

- The request is created by having the patient Alice run the FCertCon class. In this class, Alice creates her request by using the method CreateBlndSigReq(). In this method, the mobile device generates a temporal elliptical curve key pair using the EKeyGen() method. Te mobile device then generates the FCert data structure fields (i.e., the home ID, validity, and the subject). The subject is the foreign index pseudonym generated by the patient using the FIPs-Gen() method. Then, Alice generates blind factors using BlndMsgGen() and hashes the FCert data structure. Alice generates non-hierarchical home request pseudonym using nHRPs-Gen(). Then, the BlndSigReq message is constructed and MAC is generated using the HmacGen256() method. The message BlndSigReq is sent to class FCertBlndSigGen on the home server.

- Server-side verification occurs once FCertBlndSigGen class receives BlndSigReq from Alice, and the home server calls the VerifyReq() method which verifies the BlndSigReq message.

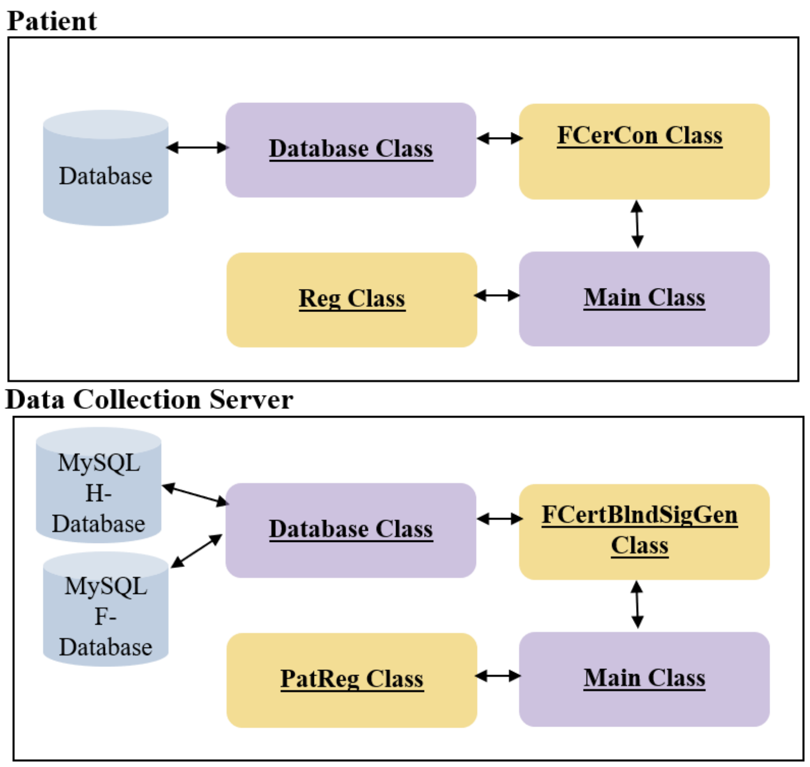


Figure A.1: Java classes to implement ADA protocols

This is done by decrypting the non-hierarchical home request pseudonym using nHRPs-Lnk() to obtain the elliptical curve public key. The home server multiplies the (EPK) with its ECC private key to get the shared key. The home shared key is used in HmacVer256() to verify the MAC of the message. The server then verifies Alice's HCert using CertVer() to ensure that it has not expired. The server also verifies the freshness of the index nonce.

- Creating the response is achieved by calling the CreateBlndSigRes() method. In this method the home server calls the BlndSigGen() and HmacGen256() methods. The BlndSigGen() method generates the blind signature, then the home server prepares the BlndSigRes message and calls the HmacGen256() to generate the MAC on the message using the home shared key. The message is then sent to the patient.

- Patient-side verification occurs once Alice receives BlndSigRes from the home server, and she uses the method VerifyRes() to verify the responnse from the server. The VerifyRes () calls the Hmac256Ver() and BlndSigDrv() methods. The Hmac256Ver() uses the home shared key to verify the MAC of the message. Then, the patient uses the BlndSigDrv() method to unblind the blind signature. The signature is then attached to the FCert.

A.1 A2DUF Prototyping and Performance Evaluation

In this section, we describe the design of the data uploading protocol for different foreign servers (A2DUF). We describe the programming language, the hardware platform, and the database, in addition to evaluating the protocol using queuing theory.

A.1.1 The Design of A2DUF Protocol

The data uploads on different foreign servers (A2DUF) consist of two participants, a patient (Alice) and a number of foreign data collection servers. The role of each entity is as follows:

- A patient: Alice performs data upload requests. To do so, she needs to (1) select one foreign server at a time for data uploads, (2) prepare her request to be sent to the selected server.
- The foreign server: Each foreign server receives the request from Alice and then performs the necessary operations, verifications, stores Alice's data, and in case of emergency sends a notification to Alice's home server.

In the following, we describe the design of the data upload on different foreign servers (A2DUF) protocol. The protocol makes use of both the hierarchical request pseudonym and non-hierarchical request pseudonym.

- Data uploads on different foreign servers using the hierarchical pseudonym: this phase describes the operations performed for data uploads by Alice and one foreign server.
  - Creating the request: This is done by the requesting patient (Alice), who creates her request message (UpReq) and sends it to the foreign server. This includes generating the data (2EMPGHD), generating the foreign request pseudonym ($\mathrm{FRP}_i^{\mathrm{f}}$), and generating the signature. Then Alice constructs the request message and finally sends the request message to the foreign server.
  - Foreign server verification: This is done by the foreign server. In this part, the foreign server receives the UpReq message from Alice and performs the necessary operations. It first uses the tag number to find the segment that contains the verification key (foreign shared key). Then, the server iterates through the keys in the segment to find the key that verified the MAC. When the server finds the key, the key is used to decrypt the foreign request pseudonym and find the foreign index pseudonym and the index nonce. The index nonce is used to ensure the freshness of the request pseudonym. Then, the foreign servers check the priority tag of the message, if it is high then the server sends a notification to Alice's home server. Otherwise, the foreign server stores the data. Finally, the server sends an acknowledgment to Alice.
  - Creating the response: This is done by the foreign server, which generates the MAC using the foreign shared key on the acknowledgment. Then, it constructs the response message (UpRes) and sends it to Alice.
  - Patient verification: This is done by Alice. When she receives the response (UpRes) from the server, Alice verifies the MAC using the foreign shared key then accepts the acknowledgment.
- Data uploads on different foreign servers using non-hierarchical pseudonyms: this describes the operations performed for data uploads by Alice and one foreign server.

- Creating the request: This is done by the requesting patient (Alice), who creates her request message (UpReq) and sends it to the foreign server. This includes generating the the data (2EMPGHD), generating the non-hierarchical foreign request pseudonym ($\mathrm{nFRP}_i^{\mathrm{f}}$), and generating a signature. Then, Alice constructs the request message and finally sends this message to the foreign server.
- Foreign server verification: This is done by the foreign server, which receives the UpReq message from Alice and performs the necessary operations. It first uses its RSA private key to decrypt the non-hierarchical foreign request pseudonym ($\mathrm{nFRP}_i^{\mathrm{f}}$). The result of the decryption is a temporal ECC public key. The foreign server multiplies the public key by its ECC private key to obtain the foreign shared key. Then, the foreign server uses the key to verify the MAC. The foreign server checks the index nonce to ensure the freshness of the request pseudonym. Subsequently, the foreign server checks the priority tag of the message. If it is high, then the foreign server sends a notification to Alice's home server. Otherwise, the foreign server stores the data. Finally, the foreign server sends an acknowledgment to Alice.
- Creating the response: This is done by the foreign server which generates the MAC using the foreign shared key on the acknowledgment. Then, it constructs the response message (UpRes) and sends it to Alice.
- Patient verification: This is done by Alice, when she receives the response (UpRes) from the server, Alice verifies MAC using the foreign shared key then accepts the acknowledgment.

A.1.2 The Implementation of A2DUF Protocol

This section describes the classes and their methods which are designed for A2DUF protocol. The protocol implementation consists of three classes: class Fupload runs on the mobile device of the patient, class Fdataupload runs on the data collection server, and class Main. Class Fupload allows the patient to create uploading requests and receives the response. Class Fdataupload allows the data collection server to receive uploading requests and creates responses. In the following, we explain these classes along with the methods used in each class.

- Class Fupload: This class runs on the patient's mobile device and it contains the following methods: CreateUpReq(), VerifyRes(), 2EMPGHD(), FRPs-Gen(), and nFRPs-Gen().
- Class Fdataupload: This class runs on a data collection server and it contains the following methods: CreateUpRes(), VerifyReq(), and FVK().
- Class Main: This class has the following methods: AesEncryption(), AeskeyGen() AesDecryption(), RsaEncryption(), RsaDecryption, HmacGen256(),HmacVer256(), Hash256(), EKeyGen(), ESigGen(), ESigVer(), getTime(), SecureRandom(), and CertVer().

The operations performed during the set-up, home and foreign registration phases are implemented as follows.

Implementation of data uploads with hierarchical request pseudonym operations:

- Creating the request: This is implemented by having the patient Alice run the Fupload class. In this class, Alice first creates her data by using the 2EMPGHD() method. This method calls a method HmacGen256() from the main class to generate a MAC using a shared key on the PGHD. This shared key is only known to the Alice and the healthcare provider (HCP). After generating the MAC, the 2EMPGHD() method calls AesEncryption() method to encrypt both the MAC and PGHD using the shared key. The result of this process is EMPGHD data. Then, the 2EMPGHD() method calls HmacGen256() to generate a second MAC on EMPGHD using the home shared key. After that the 2EMPGHD() method calls the AesEncryption() method to encrypt both the second MAC and EMPGHD. The result of this process is 2EMPGHD data. Second, Alice uses the CreateUpReq() method, which calls a FRP-Gen() method to generate the foreign request pseudonym. Then, CreateUpRe() constructs the UpReq message and generates MAC on the message using HmacGen256() and the foreign shared key. The message UpReq is sent to class FDataUpload on the foreign server.

- Server-side verification: Once the foreign server receives UpReq from Alice, the foreign server calls VerifyReq() method, which uses the tag number as an input to the FVS() method. This method is used to find the foreign shared key and verify the message using the foreign shared key. The FVS() method uses the tag number to locate the segment. It then iterates through the segment and uses each key to verify the MAC. Then, the key which verified the MAC is used as an input to the FRPs-Lnk() to find the foreign index pseudonym and the index nonce to ensure the freshness of the request pseudonym. The foreign index pseudonym is used to retrieve the FCert. The server uses the CertVer() method from Main class to ensure that the FCert is within 24 hours and has not expired. Then FDataUpload calls a PrChecker() to check the priority of the message. If the priority is high the FDataUpload sends notification to Alice's home server. Otherwise, the FDataUpload class calls InsertData() to insert the 2EMPGHD into the F-DataBase.

- Creating the response: This achieved by calling the CreateUpRes() method. In this method the foreign server calls HmacGen256() to generate the MAC on the acknowledgement using the foreign shared key. The message is then sent to the patient.

- Patient-side verification: Once Alice receives UpRes from the foreign server, she uses the VerifyRes() method to verify the response from the server. The VerifyRes() method calls the HmacVer256() which uses the foreign shared key to verify the MAC of the message.

Implementation of data uploads using non-hierarchical request pseudonym operations:

- Creating the request: This is implemented by having the patient Alice run the Fupload class. In this class, Alice first creates her data by using the 2EMPGHD() method. This method calls the HmacGen256() method from the main class to generate a MAC using a shared key on the PGHD. This shared key is only known to the Alice and the healthcare provider (HCP). After generating the MAC, the 2EMPGHD() method calls the AesEncryption() method to encrypt both the MAC and PGHD using the shared key. The result of this process is EMPGHD data. Then, the 2EMPGHD() method calls HmacGen256() to generate a second MAC on EMPGHD using the home shared key. After that the 2EMPGHD() method calls the AesEncryption() method to encrypt both the second MAC and EMPGHD. The result of this process is 2EMPGHD data. Second, Alice uses the CreateUpReq() method which calls the nFRP-Gen() method to generate the non-hierarchical foreign request pseudonym. Then, CreateUpRe() constructs the UpReq message and generates MAC on the message using HmacGen256() and the foreign shared key. The UpReq message is sent to class FDataUpload on the foreign server.

- Server-side verification: Once the foreign server receives UpReq from Alice, it uses the VerifyReq() method which calls RsaDecryption() method to decrypt the non-hierarchical foreign request pseudonym and discover the temporal ECC public key. Then, the server multiplies its ECC private key with the obtained public key to obtain the foreign shared key. Then, the key is used to verify the MAC associated with the message. The server then checks both the index nonce to ensure the freshness of the request pseudonym, and the FCert using CertVer() to ensure that the FCert is within 24 hours and has not expired. Then FDataUpload calls a PrChecker() to check the priority of the message. If the priority is high, the FDataUpload sends notification to Alice's home server. Otherwise, the FDataUpload class calls InsertData() to insert the 2EMPGHD into the F-DataBase.

- Creating the response: This is achieved by calling the CreateUpRes() method. In this method the foreign server calls HmacGen256() to generate the MAC on the acknowledgement using the foreign shared key. The message is then sent to the patient.

- Patient-side verification: Once Alice receives UpRes from the foreign server, she uses the VerifyRes() method to verify the response from the server. The VerifyRes() method calls the HmacVer256() method, which uses the foreign shared key to verify the MAC of the message.

Appendix B

Benchmarking

This appendix presents some of benchmarking codes that we used for the performance evaluation of the protocols presented in this thesis.

B.1 Java Microbenchmark Harness (JMH)

The typical method which is used to measure the execution time of a protocol relies on running a protocol a few times in a loop and measuring the elapsed time with a stopwatch, System.currentTimeMillis(), or System.nanoTime() around method invocation. This typical method does not give an accurate measurement of the execution time of the protocol. This is because the Java virtual machine (JVM) is continuously optimizing the bytecode. This optimization invalidates the performance model. Thus, to get an accurate performance measurement we use a Java benchmarking tool called Java Microbenchmark Harness (JMH). The JMH is developed by the same developer who implements the Java virtual machine, this makes it an excellent tool to be used to benchmark the Java application. We use the JMH tool to measure the execution of each cryptographic method used in our protocol separately. The summation of the execution time for all methods used in the protocol is the overall execution time for the protocol. This way we ensure that the precise measurement of the execution time of the protocol.

The JMH requires the use of some annotations at a class level and a method level. Regarding the class level, we use the annotations Warmup, Fork, and Benchmark Mode. The Warmup annotation helps to specify the number of warmup iterations before running the benchmarked forks. The warmup helps JVM to perform any class loading, compilation to native code, and caching steps. It would normally apply in a long-running application before starting to collect actual results. The warmup iterations are ignored when creating the benchmark results. The fork annotation helps to specify the number of iterations needed before getting the actual result. The measurement annotation allows users to determine what they want to measure. It has several modes, one of which is the average time. The average time measures the average time it takes for one's benchmark method to be executed. For the method level, we use benchmark annotation to mark the method for which we want to measure the execution time. We place the annotation before the beginning of the method.

Table 6.2 below shows the time in ms for each method used in the inter-domain (HSR and

FSR) protocols. The table shows the number of cryptographic methods used by each entity that participates in the protocol, the key size used for each method, and the time in ms required to execute each method. Each value in this table represents the mean value of 20 forks. For each fork, we ran 20 warm-up iterations. The value of warm-up iterations is ignored. After the 20 warm-up iterations, we ran 20 benchmark iterations. The same is true for Table 6.3 and Table 6.4 which show the time required to execute each method used in intra-domain protocols by both sides. The coding details are given in Appendix A.

B.2 Benchmarking Codes

B.2.1 CryptoBase Class

```
@Fork(20)
@Warmup(iterations = 10, time = 1000, timeUnit = TimeUnit.MILLISECONDS)
@Measurement(iterations = 10, time = 1000, timeUnit = TimeUnit.MILLISECONDS)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
public class CryptoBase {
    @Param({""})
    private String provider;
    public Provider prov = null;

    @Setup
    public void setupProvider() {

        if (provider != null && !provider.isEmpty()) {

            prov = Security.getProvider(provider);

            if (prov == null) {
                throw new RuntimeException
                ("Can't find prodiver \"" + provider + "\"");

            }
        }
    }

    public static Cipher makeCipher(Provider prov, String algorithm)  {
        return (prov == null) ? Cipher.getInstance(algorithm) :
                Cipher.getInstance(algorithm, prov);

    }

    public static byte[][] fillRandom(byte[][] data) {
        Random rnd = new Random();
        for (byte[] d : data) {
            rnd.nextBytes(d);
        }
        return data;
    }

    public static byte[] fillRandom(byte[] data) {
        Random rnd = new Random();
        rnd.nextBytes(data);
        return data;
    }

    public static byte[] fillSecureRandom(byte[] data) {
        SecureRandom rnd = new SecureRandom();
        rnd.nextBytes(data);
        return data;
    }

    public static byte[][] fillEncrypted(byte[][] data, Cipher encryptCipher)

        byte[][] encryptedData = new byte[data.length][];
        for (int i = 0; i < encryptedData.length; i++) {
            encryptedData[i] = encryptCipher.doFinal(data[i]);
        }
        return encryptedData;
    }

}
```

B.2.2 AES Class

```
public class AESBench extends CryptoBase {
    public static final int SET_SIZE = 128;
```

```
@Param({"AES/CBC/PKCS5Padding"})
private String algorithm2;
@Param({"256"})
private int keyLength;
@Param({"3000000"})
private int dataSize;
byte[][] data;
byte[][] encryptedData;
private Cipher encryptCipher;
private Cipher decryptCipher;
int index = 0;

@Param({"HmacSHA256"})
private String algorithm;

private Mac mac;

@Setup
public void setup()  {
 setupProvider();

byte[] keystring = fillSecureRandom(new byte[keyLength / 8]);
SecretKeySpec ks = new SecretKeySpec(keystring, "AES");
encryptCipher = makeCipher(prov, algorithm2);
 encryptCipher.init(Cipher.ENCRYPT_MODE, ks);
decryptCipher = makeCipher(prov, algorithm2);
decryptCipher.init
 (Cipher.DECRYPT_MODE, ks, encryptCipher.getParameters());
 data = fillRandom(new byte[SET_SIZE][dataSize]);
 encryptedData = fillEncrypted(data, encryptCipher);
}
@Benchmark
public byte[] encrypt()  {
    byte[] d = data[index];
    index = (index +1) % SET_SIZE;
   return encryptCipher.doFinal(d);
}
@Benchmark
public byte[] decrypt()  {
    byte[] e = encryptedData[index];
    index = (index +1) % SET_SIZE;
    return decryptCipher.doFinal(e);
    }

public static void main(String[] args) {
    Options opt = new OptionsBuilder()
        .include(AESBench.class.getSimpleName())
        .build();

    new Runner(opt).run();
}




}
```

B.2.3 RSA Class

```
public class RSABench extends CryptoBase {

    public static final int SET_SIZE = 128;
    @Param({ "RSA/ECB/PKCS1Padding"})
    protected String algorithm;
    @Param({"200"})
    protected int dataSize;
    @Param({"15360"})
    protected int keyLength;
    private byte[][] data;
    private byte[][] encryptedData;
    private Cipher encryptCipher;
    private Cipher decryptCipher;
    private int index = 0;

    @Setup()
    public void setup()  {
        setupProvider();
        data = fillRandom(new byte[SET_SIZE][dataSize]);
        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
        kpg.initialize(keyLength);
        KeyPair keyPair = kpg.generateKeyPair();
        encryptCipher = makeCipher(prov, algorithm);
        encryptCipher.init
        (Cipher.ENCRYPT_MODE, keyPair.getPublic());
         decryptCipher = makeCipher(prov, algorithm);
         decryptCipher.init
        (Cipher.DECRYPT_MODE, keyPair.getPrivate());
         encryptedData = fillEncrypted(data, encryptCipher);
    }

    @Benchmark
    public byte[] encrypt()  {
        byte[] d = data[index];
        index = (index +1) % SET_SIZE;
        return encryptCipher.doFinal(d);
    }

    @Benchmark
    public byte[] decrypt() throws  {

        byte[] e = encryptedData[index];
        index = (index +1) % SET_SIZE;
        return decryptCipher.doFinal(e);
    }


    public static class Extra extends RSABench {
        @Param({"RSA/ECB/PKCS1Padding"})
        private String algorithm;
        @Param({"200"})
        private int dataSize;
        @Param({"2048", "15360"})
        private int keyLength;
    }
    public static void main(String[] args)  {
    Options opt = new OptionsBuilder()
        .include(RSABench.class.getSimpleName())
        .build();
    new Runner(opt).run();
}




}
```

B.2.4 HMAC Class

```
public class MacBench extends CryptoBase {
    public static final int SET_SIZE = 128;
    @Param({"HmacSHA256"})
    private String algorithm;
    @Param({"351856"})
    int dataSize;
    private byte[][] data;
    private Mac mac;
    int index = 0;
    @Setup
    public void setup() throws  {
      setupProvider();
        mac = (prov == null) ?
       Mac.getInstance(algorithm) : Mac.getInstance(algorithm, prov);
        mac.init(KeyGenerator.getInstance(algorithm).generateKey());
        data = fillRandom(new byte[SET_SIZE][dataSize]);
    }
    @Benchmark
    public byte[] mac() {
        byte[] d = data[index];
        index = (index + 1) % SET_SIZE;
        return mac.doFinal(d);

    }
}
```

B.2.5 Digest Class

```
public class MessageDigestBench extends CryptoBase {
    public static final int SET_SIZE = 128;
    @Param({"SHA-256"})
    private String algorithm;


    @Param({""+1024*1024})
    int dataSize;
    private byte[][] data;
    int index = 0;
    @Setup
    public void setup() {
        setupProvider();
        data = fillRandom(new byte[SET_SIZE][dataSize]);
    }

    @Benchmark
    public byte[] digest()  {
   MessageDigest md = (prov == null) ?
MessageDigest.getInstance(algorithm) :
MessageDigest.getInstance(algorithm, prov);
        byte[] d = data[index];
        index = (index +1) % SET_SIZE;
        return md.digest(d);

    }




}
```

B.2.6 SQL Class

```
public class MySQLBenchmark extends CryptoBase{
public static Connection conn;
   public static final int SET_SIZE = 128;
    @Param({ "SHA-256"})
    private String algorithm;
    @Param({"700"})
    int dataSize;
    private byte[][] data;
    int index = 0;
@Setup
    public void setup() throws SQLException, Exception {
String myDriver = "com.mysql.cj.jdbc.Driver";
     String myUrl = "jdbc:mysql://localhost/fdcsrvdb";
Class.forName(myDriver)
conn = DriverManager.getConnection(myUrl, "root", "root");
setupProvider();
```

```
data = fillRandom(new byte[SET_SIZE][dataSize]);
    }
//------------------------------------------------------------------
    @TearDown
    public  void  tearDown() {
     conn.close();
    }
//===================================================================
    @Benchmark
    public String select() {
      String hsk=null;
      String query = "SELECT id,author_id,title
           FROM posts where id between 1 and 900";
      Statement st = conn.createStatement();
      ResultSet rs = st.executeQuery(query);

      while (rs.next())  {
       digest() ;
      }
         return hsk;
    }

    //-------------------
    @Benchmark
    public byte[] digest()  {
        MessageDigest md = (prov == null) ?
MessageDigest.getInstance(algorithm) :
MessageDigest.getInstance(algorithm, prov);
             byte[] d = data[index];
             index = (index +1) % SET_SIZE;
             return md.digest(d);

         }

   @Benchmark
    public String selectCert(){

     String sql = "select cer,hip FROM pcert  where hip="+ name;
        String selected = null;
        try (
             Statement stmt  = conn.createStatement();
             ResultSet rs    = stmt.executeQuery(sql)){
```

```
                            // loop through the result set
                 selected=    rs.getString("cer") ;
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }

        return selected;
    }

   @Benchmark
    public void insert( ) {

        String sql = "INSERT INTO pcert(hip,pr,pk,cer) VALUES(?,?,?,?)";

        try (
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, hip);
            pstmt.setString(2, pr);
            pstmt.setBytes(3, pk);
            pstmt.setString(4, cer);
            pstmt.executeUpdate();
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }




}
```

B.2.7 Signature Class

```
public class SignatureBench extends CryptoBase {

    public static final int SET_SIZE = 128;
    @Param({"SHA256withECDSA"})
    private String algorithm;
    @Param({"2000"})
    int dataSize;
    @Param({"256"})
    private int keyLength;
    private PrivateKey privateKey;
    private PublicKey publicKey;
    private byte[][] data;
    private byte[][] signedData;

    int index;

    private String getKeyPairGeneratorName() {
        String tail = algorithm.substring(algorithm.lastIndexOf("with") + 4);
        return "ECDSA".equals(tail) ? "EC" : tail;
    }

    @Setup()
    public void setup()  {
        setupProvider();
        KeyPairGenerator kpg =
KeyPairGenerator.getInstance(getKeyPairGeneratorName());
        kpg.initialize(keyLength);
        KeyPair keys = kpg.generateKeyPair();
        this.privateKey = keys.getPrivate();
        this.publicKey = keys.getPublic();
        data = fillRandom(new byte[SET_SIZE][dataSize]);
        signedData = new byte[data.length][];
        for (int i = 0; i < data.length; i++) {
            signedData[i] = sign(data[i]);

        }
    }

    public byte[] sign(byte[] data) {
        Signature signature = (prov == null) ?
Signature.getInstance(algorithm) :
Signature.getInstance(algorithm, prov);
        signature.initSign(privateKey);
        signature.update(data);
        return signature.sign();
    }

    @Benchmark
    public byte[] sign() {
        byte[] d = data[index];
        index = (index + 1) % SET_SIZE;
        return sign(d);
    }

    @Benchmark
    public boolean verify()  {
        Signature signature = (prov == null) ?
Signature.getInstance(algorithm) :
Signature.getInstance(algorithm, prov);
        signature.initVerify(publicKey);
        byte[] d = data[index];
        byte[] s = signedData[index];
        index = (index + 1) % SET_SIZE;
        signature.update(d);
        return signature.verify(s);
    }

   public static class RSA extends SignatureBench {
        @Param({"MD5withRSA", "SHA256withRSA"})
        private String algorithm;
        @Param({"1024", "2048", "3072"})
        private int keyLength;
    }

    public static class ECDSA extends SignatureBench {
        @Param({"SHA256withECDSA"})
        private String algorithm;
        @Param({"256"})
        private int keyLength;
    }
    public static void main(String[] args)
 {
    Options opt = new OptionsBuilder()
        .include(SignatureBench.class.getSimpleName())
        .build();
    new Runner(opt).run();
}
}
```

B.2.8 Blind Signature Class

```
public class BlindSigniture{



    public static void main (String[] args)
    {
     Options opt = new OptionsBuilder()
                .include(BlindSigniture.class.getSimpleName())
                .forks(1)
                .build();
        new Runner(opt).run();

    }
    @Setup
    public static  void setup()
{
   Security.addProvider(new BouncyCastleProvider());
// ka = KeyAgreement.getInstance("ECDH","BC");
         curveParameters = ECUtil.getNamedCurveByName("secp256k1");
         genPoint = curveParameters.getG();
         n = curveParameters.getN();
         ecParameterSpec = new ECParameterSpec(curveParameters.getCurve(),
                  genPoint, curveParameters.getN());
         keyPairGenerator = KeyPairGenerator.getInstance("EC","BC");
        keyPairGenerator.initialize(ecParameterSpec);
        random = new SecureRandom();
         keyPair = keyPairGenerator.generateKeyPair();
          pk = keyPair.getPublic();
          sk = keyPair.getPrivate();
          pkPoint = ((ECPublicKey) pk).getQ();
          d = ((ECPrivateKey) sk).getD();
          k = getNextRandomBigInteger(random,32);
          rPoint = genPoint.multiply(k);
         alpha = getNextRandomBigInteger(random,32);
         beta = getNextRandomBigInteger(random,32);
         ceta = getNextRandomBigInteger(random,32);
         data = fillRandom(new byte[90]);
        Client_init ();
        cal();

}
    @Benchmark
public static    void Client_init ()
```

```
{
       // byte[] data = {1, 2, 2, 3, 4, 5, 6, 7, 8, 9, 10};
         blinddata=  blind_data();
         sig = signer ();
         unblndsig =  unblinding();
        boolean b= verifier();
     //   System.out.println(b);
}
//=========================================================================
    @Benchmark
    public static   BigInteger blind_data( )
    {



         aG = genPoint.multiply(alpha);
         bQ = pkPoint.multiply(beta);
         aPoint = rPoint.add(aG).add(bQ);
    //----------- R=aR+cP-bY -----------------------
         aRp = rPoint.multiply(alpha);
         cP = genPoint.multiply(ceta);
         by = pkPoint.multiply(beta);
         cPaRp= aRp.add(cP).subtract(by);
    //------------------e=H(R||m)---------------------
         t2 = getX(cPaRp).mod(n);
      t2Byte = t2.toByteArray();
         e = hashCombine(data, t2Byte);
         ee = new BigInteger(e);
    //-------------------ee= a-1 (e-b)mod n------------------------
         eeb= ee.add(beta.negate());
         aa= alpha.modInverse(n);
          aaa=aa.multiply(eeb);
    //    System.out.println("execute 3");
        return aaa;

    }

    //-----------------------------------------------------------------------
    @Benchmark
    public static BigInteger unblinding ( )
    {
 //--------------S= Sa+cmod n ---------------------
      sigalpha=  sig.multiply(alpha);
```

```
       muhgh = ceta.mod(n);
      verify = sigalpha.add(muhgh);



     return verify;
    }
//=======================================================
    @Benchmark
    public  static BigInteger signer(   )
    {
        //------------------s=eSpr+Kp mod n------------------------
         eS= blinddata.multiply(d);
         ess= k.mod(n);
         sig2 =eS.add(ess);

        return sig2;
    }
  //---------------------------------------------------------
    @Benchmark
  public static    boolean verifier ()
    {
     byte[] array = unblndsig.toByteArray();
     if (array[0] == 0) {
         byte[] tmp = new byte[array.length - 1];
         System.arraycopy(array, 1, tmp, 0, tmp.length);
         array = tmp;
     }

      SPP = genPoint.multiply(unblndsig);
      ey= pkPoint.multiply(ee);
      sppey=SPP.add(ey.negate());
      tttt = getX(sppey).mod(n);
    tttByte = tttt.toByteArray();
cvvByte = hashCombine(data, tttByte);
      cvvv = new BigInteger(cvvByte);

     return cvvv.equals(ee);
    }
//------------------------------------------------------------------------------
    @Benchmark
public static void cal()
{
  rPoint = genPoint.multiply(k);
}
  public static byte[] fillRandom(byte[] data) {
        Random rnd = new Random();
        rnd.nextBytes(data);
        return data;
    }
public static  byte[] hashCombine(byte[] data1, byte[] data2) {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        digest.update(data1);
        digest.update(data2);
        return digest.digest();
    }
//---------------------------------------------------------------------------------

    public static  BigInteger getNextRandomBigInteger(Random random, int byteSi
        byte[] byteArray = new byte[byteSize];
        random.nextBytes(byteArray);
        return new BigInteger(byteArray);
    }

    public static BigInteger getX(ECPoint point) {
        return point.normalize().getXCoord().toBigInteger();
    }
//---------------------------------------------------------------------------------


}
```