Design and Implementation of a Blockchain-based Global Authentication System Using Biometrics and Subscriber Identification Module

Navid Khalili

Thesis submitted in partial fulfillment of the requirements for the
Master of Applied Science degree in Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

The digital world tolerates a high volume of information and interactions. Considering the usage of electronic services by authorities, User Authentication (UA) is crucial. Numerous authentication methods are proposed in the literature; yet, identifying users based on their actual identities with the capability of global usage and respecting privacy is under research. By adopting Blockchain technology in the software industry, the record management systems have satisfied properties such as transparency, accountability, anonymity, and attack resiliency. Moreover, Smartphones are powerful devices capable of hosting a Subscriber Identification Module (SIM) Card that secures the execution of processes involving use of sensitive information. A combination of these technologies is the foundation of a strong UA in cyberspace. In this thesis, we propose the design and prototype of Blockchain-based Global Authentication System (BBGAS) that offers a secure, privacy-preserving, and transparent authentication system based on users' biometrics via Smartphones appropriate for service provider applications.

Acknowledgments

I would like to express my sincere gratitude to my supervisor Prof. Dimitrios Makrakis whose support and motivation shed light on my journey, especially throughout the roughest moments of my life. Without his guidance, it would have been unfeasible to complete my degree. My appreciation also goes to Prof. Abdelhakim Senhaji Hafid from the University of Montreal, who co-supervised my work during the course of my thesis. His comments and advice have continuously strengthened my knowledge towards the completion of my work. I am also very appreciative of the financial aid I received from my supervisors.

My thoughts also go to my beloved family; my parents, who have always been supportive and encouraging both financially and emotionally during my entire life, particularly during the challenging years of the COVID-19 pandemic; my sister, who has constantly been an inspiration in my life; as well as my brothers who have spent adequate time in listening to me and offering solutions to my problems.

Similarly, I would like to thank my friends in Ottawa who have supported me during my stay in Canada, including Mr. Arash Azarnoush, my roommate and best friend who had been there when needed.

I am also grateful to the University of Ottawa staff and my thesis committee for spending their precious time evaluating my thesis work. I am also thankful for the International Graduate Bursary received from the University of Ottawa, their Teaching Assistant positions and the CUPE union's support.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition
AAM	Application Abstract Machine
AES	Advanced Encryption Standard
ANS	Answer
API	Application Programming Interface
BA	Byzantine Agreement
BBGAS	Blockchain-based Global Authentication System
BC	Blockchain or Blockchain Network
BF	Byzantine Fault
BFS	BFT distributed File System
BFT	Byzantine Fault Tolerance
BRSD	Biometric Recording and Storing Devices
CA	Certificate Authority
CAD	Card Access Device
CHAP	Challenge Handshake Authentication Protocol
CLI	Command-line Interface
CoBs	Chain of Blocks
DES	Data Encryption Standard
DFA	Differential Fault Analysis
DH	Diffie–Hellman (Key exchange)
DLTs	Distributed Ledger Technologies

DMA	Direct Memory Access
DPA	Differential Power Analysis
DRF	Deterministic Reproduction Function (a.k.a. Rep(.))
DSA	Digital Signature Algorithm
ECC	Elliptic Curved Cryptography
ECDH	Elliptic-curve Diffie–Hellman (Key exchange)
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
EEPROM	Electrically Erasable Programmable Read-only Memory
E-Gov	Electronic Government
e-ID	Electronic Identity Card
e-Service	Electronic Service
eSIM	Embedded SIM
FCC	Fabric CA Client
FPS	Fingerprint Sensor
FAPS	Fake Prevention System
GAS	Global Authentication System
GB	Genesis Block
GC	Garbage Collector
GIDSP	Global Identification Service Provider (a.k.a. SP)
GIDSPL	Global Identification Service Provider's Location
HAL	Hardware Abstraction Layer
HIDL	Hardware Interface Definition Language

HMAC	Hash-based Message Authentication Code
HO	Health Organization
ICT	Information and Communications Technology
ID	Identification, Identity
IoT	Internet-of-Things
ISIM Card	Intelligent Subscriber Interface Module Card
ISIMC	Intelligent Subscriber Interface Module Card
IVP	Identity Verification Provider
JVM	Java Virtual Machine
KB	Kilobyte
LCR	Longest Chain Rule
MAA	Modular Arithmetic API
MD	Message-digest algorithm
MEL	MULTOS Executable Language
MITM	Man-in-the-middle attack
MULTOS	Multi-application Smart Card Operating System
NAND	Non-volatile memory
NFC	Near-field Communication
NIST	National Institute of Standards and Technology
NW	Normal World
OS	Operating System
OTP	One-Time Password
P2P	Peer-to-Peer

PAP	Password Authentication Protocol
PBFT	Practical Byzantine Fault Tolerance
PGF	Probabilistic Generation Function (a.k.a. Gen(.))
PIN	Personal Identification Number
PKI	Public Key Infrastructure
PNs	Permissioned Nodes
PWD	Password
QST	Question
RAM	Random Access Memory
REE	Rich Execution Environment
RFID	Radio Frequency Identification
RND	A function to generate a random integer number
RNG	Random Generator Number
ROM	Read-only Memory
RP	Relying Party
RSA	Rivest-Shamir-Adleman
SCP	Stellar Consensus Protocol
SD	Smart Device
SELinux	Security-Enhanced Linux (SELinux)
SGX	Intel Software Guard Extensions
SHA	Secure Hash Algorithm
SIM	Subscriber Identification Module
SIN	Social Insurance Number

SMC	Secure Monitor
SO	Service Organization
SO2SP	ServiceOrganization-to-ServiceProvider communication
SP	Service Provider (a.k.a. GIDSP)
SPA	Simple Power Analysis
SSL	Secure Sockets Layer
SW	Secure World
TA	Trusted Application
TDES	Triple Data Encryption Standard
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TP	Trust Provider
TRNG	True Random Number Generator
Tx(s)	Transaction(s)
U2SO	User-to-ServiceOrganization communication
U2SP	User-to-ServiceProvider communication
U2U	User-to-User communication
UAID	User Authentication ID
UART	Universal Asynchronous Receiver Transmitter
UI	User Interface
UR	User
VMM	Virtual Machine Monitor
WSL	Windows Subsystem for Linux

XOR Exclusive-OR

List of Symbols

Symbol	Definition
$\boldsymbol{GIDSP}$	Global Identification Service Provider
$\boldsymbol{SO}$	Service Organization
$\boldsymbol{s_{GIDSP}}$	Private/Secret key of GIDSP
$\boldsymbol{Pub_{GIDSP}}$	Public key of GIDSP
$\boldsymbol{SRV_j}$	j^{th} server of SO
$\boldsymbol{\alpha_{SRV_j}}$	j^{th} server additional data (e.g., name, type)
$\boldsymbol{ID_{SRV_j}}$	Identity of j^{th} server of SO
$\boldsymbol{Pub_{SRV_j}}$	Public key of j^{th} server of SO
$\boldsymbol{UR_i}$	i^{th} user
$\boldsymbol{ID_{UR_i}}$	i^{th} user real identity
$\boldsymbol{PID_{UR_i}}$	i^{th} user public identity
$\boldsymbol{PW_{UR_i}}$	User's i^{th} password
$\boldsymbol{G}$	Elliptic curve points group
$\boldsymbol{P}$	Generator of G
$\boldsymbol{SK_{i,j}}$	Session-key between i and j
$\boldsymbol{\xi_{UR_i}}$	Encrypted version of questions for the i^{th} user
$\boldsymbol{v_{UR_i}}$	Encrypted version of answers for the i^{th} user
$\boldsymbol{Enroll_{BC}}(.)$	Blockchain CA's enrollment function
$\boldsymbol{Enc}(.)$	Encryption function with the provided public key

$\Rightarrow$	Secure channel
$\rightarrow$	Public channel
$\oplus$	XOR operation
$\|\|$	Concatenation
$\boldsymbol{H_i}(.)$	Hash function ($0 \leq i \leq 6$)
$\boldsymbol{Bio_{UR_i}}$	User i's biometric
$\boldsymbol{Gen}(.)$	Probabilistic generation used by fuzzy extractor
$\boldsymbol{Rep}(.)$	Deterministic reproduction
$\boldsymbol{k_i}$	Biometric i's secret key
$\boldsymbol{\tau_i}$	Biometric i's public reproduction parameter
$\boldsymbol{BID_{UR_i,SRV_j}}$	i^{th} user or j^{th} server blockchain identity
$\boldsymbol{Data_{UR_i,SRV_j}}$	i^{th}/j^{th} user/SOs' registration data
$\boldsymbol{PID_{SRV_j}}$	Public Identity of j^{th} server
$\boldsymbol{RN_{SRV_j}}$	Request Ticket Number generated by j^{th} server
$\boldsymbol{T_{SRV_j}}$	Request Ticket Timestamp generated by j^{th} server
$\boldsymbol{Info_{SRV_j}}$	Additional info of j^{th} server
$\boldsymbol{PID_{UR_i}}$	Public Identity of i^{th} user
$\boldsymbol{AN_{UR_i}}$	Accept Ticket Number generated by i^{th} user
$\boldsymbol{T_{UR_i}}$	Timestamp of the acceptance message generated by i^{th} user
$\boldsymbol{Info_{UR_i}}$	Additional info of i^{th} user

Chapter 1 - Introduction

1.1. Background

1.1.1. Authentication Techniques

The proliferation of software applications has seen significant increases as our technology advances at high speed. Many organizations develop software applications to improve customer experience and adopt automation approaches to accelerate their procedures. As such, software systems must protect users' information at a very preliminary step by first authenticating then authorizing a user who has requested a specific service, aside from the security concerns of the data. In this context, three tightly coupled definitions are involved: (1) Identification, (2) Authentication, and (3) Authorization. First, a document that a user provides to an organization to assert his identity is called Identification; for example, a driver's license is a physical identification used by governments. Then, the recognition process that the organization conducts with the help of the mentioned document is called Authentication. Finally, the checking processing of the user's privilege (a.k.a. Access Control) by the organization to reveal the protected resources is called Authorization [1]. It is noteworthy that this manuscript works around the two first definitions, (1) and (2); therefore, the third definition is beyond the scope of this document.

In the Identification and Authentication of a user, these questions must be answered: (A) who is the person that requested a service? And (B) is he/she the real person that he/she claims to be? The precondition to finding the answer to these questions is Registration. Without having the user enrolled in the system, it is impossible to authenticate the user. The result of the registration phase is to produce information that can be given to the user for future communication. This information in software systems is included but not limited to:

- **Passwords:** combinations of strings, easy or hard to memorize,
- **USB tokens:** a piece of hardware with the information or certificate,
- **PIN codes:** it is the same as passwords but in numerical values,
- **RFID cards:** use RFID tags to be identified by readers
- **Biometrics:** samples of fingerprints or users' images.

1.1.2. Data Security and Cryptography

Although these techniques are beneficial in specific scenarios, they can be stolen or compromised. The first countermeasure to reduce the risk of stealing identities by attackers is the usage of multi-factor authentication [2]. These factors can be a combination of a user's knowledge, possession, and characteristics. An Example of each factor is Password, Smart Card, and Biometrics, respectively. The second way to keep the identifications safe is to protect the communication between two parties. But before that, it is essential to know that the connection between a user and an authentication system is known as a Channel [1]. A channel can be one of the following types:

- **Authentic:** this channel is tamper-resistant,
- **Confidential:** it is interception resistant,
- **Secure:** it is both authentic and confidential,
- **Insecure:** this type has none of the above-listed channel types.

Sending a plain identification over the communication channel in Password Authentication Protocol (PAP) is known as Static Authentication, which has many drawbacks. The best-known attack against this method is when a third-party listens to the communication channel. The eavesdropper can use the information to obtain access to the system and performs a Replay attack [3]. Moreover, this method is also susceptible to physical attacks like a recording camera or a

person next to the user watching when entering the password. It is also possible for malicious software/hardware to act as a keylogger to record the information. On the other hand, the Challenge Handshake Authentication Protocol (CHAP) sends a hashed version of the identification for a challenge/response scenario that still is vulnerable to Brute-force and Dictionary attacks [4]. Thus, Cryptography helps mitigate these attacks by using either a shared secret key or public-key cryptography in challenge-based scenarios. However, it is essential to keep the shared or private key confidential while performing the encryption and decryption operations. In addition, utilizing Smart Cards, in such sensitive scenarios, provides a secure and isolated execution environment to perform cryptographic processes. Smart Cards can also verify the legitimate user by performing PIN, biometric, or password authentication prior to the calling procedure locally [1].

1.1.3. Electronic Governments

Identification and Authentication are even more crucial in smart cities where Electronic Governments (E-Govs) are growing. Various organizations in a smart city cooperate to offer services that are sustainable, efficient, and of high quality (a.k.a. urbanization). The organizations focus on concepts such as Mobility, Economy, Environment, Living, People, and Governance in the smart city context. *Mobility* dimension concentrates on means of transportation; *Economy* centers innovations and collaboration of enterprises; *Environment* emphasizes energy consumption and environmental activities; *Living* targets the citizens' quality of life such as healthcare and housing; *People* involves education towards social capital and ethnic society; and finally *Governance* delivers Electronic Services (e-Services) to empower decision makings [5]. E-Govs point to the Governance definition in smart cities that moves at fast pace to global adoption. This adoption is achievable by the rapid development of Information and Communications Technology (ICT) such as Cloud Computing, Machine Learning, Internet-of-Things (IoT), and

Big Data [6]. Therefore, the importance of the users' identities and accordingly the authentication procedure of citizens are quite evident in e-Services introduced to the public.

1.1.4. Smartphones

The Smartphones (SPs) industry followed the same trend as software applications. SPs have been improved both hardware-wise and software-wise. Manufacturers equip SPs with powerful resources such as processors, sensors, and memories. Similarly, they have been adopting various kinds of biometric sensors to their devices. At first, biometrics were used to unlock the phones instead of using password. This approach provides fast, convenient, and accurate authentication for users compared to manually entered PINs [7]. The idea even goes further by designing a biometric Application Programming Interface (API) for SPs to provide authentication services to third-party applications [8]. This feature, in modern devices, does not expose any biometric information to external applications and the process happens inside a secure environment isolated from the rest of the phone for security merits. When a desktop mobile application sends a new authentication request to the biometric system via this API, a dialog box appears on the screen asking the user to provide his biometrics to the SP's sensor. After providing the biometric identification by the user, the matching process starts, and the API only returns the result of authentication in a Boolean value indicating whether the authentication was successful or unsuccessful. By doing so, the third-party application does not require a separate authentication system and it utilizes the implemented method by the phone manufacturer that also benefits from biometrics. It is worth mentioning that this API does not expose any information to neither the outside environment such as Cloud, nor to other parts of the device that may host malicious applications [9].

According to [10], the worldwide market share of popular operating systems on June 2021 is approximately 99.18% in total, with the share of Android about 72.84% and iOS around 26.34%. In addition, the Smartphone global market share of Samsung was close to 22% and Apple's roughly 16% in Q1 of 2021 [11]. Taking these two statistics into account, we describe and target these two OSs and devices in this thesis to examine their biometrics and execution environments. Starting with Samsung, it introduced KNOX in 2013 as its secure platform with the Galaxy series phones [12]. The goal of this platform was to provide a tamper-resistant environment for secure processing on Samsung phones for applications such as Biometrics Authentication. Similarly, Android and iOS provide a secure solution (see Chapter 3) for sensitive operations. Similar to KNOX, Android utilizes ARM TrustZone technology for implementing this environment [8]. TrustZone divides the physical system into two independent environments with diverse levels of execution known as Normal World (NW) and Secure World (SW). The former runs the Rich Execution Environment (REE) functionalities that operates on Rich OS, while the latter is designed for security-backed services that processes sensitive data running on the Secure OS (see *Figure 1).* Although SW has the privilege to access NW completely and utilize SW partially, NW has no access to SW. Hence, SW is protected from malicious applications which are running in the less secure environment (a.k.a. NW) [12].

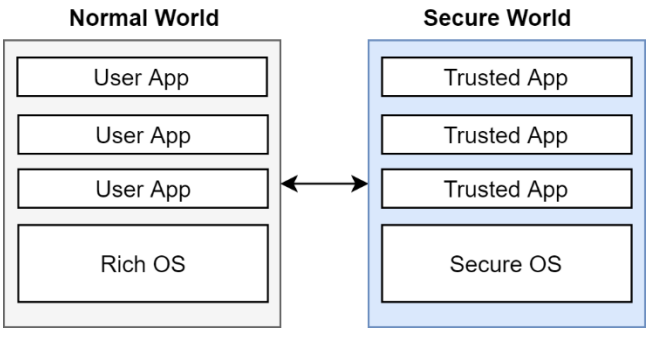


Figure 1 – Simplified version of Normal Word and Secure World [12]

1.1.5. Blockchains

Data storage and record change management have also improved during the past decades. With introducing Bitcoin [13] in 2008, the concept of Blockchain (BC) emerged. This concept utilized the timestamp concept that was first introduced in 1991 for determining the originality of a digital document [14]. Noteworthy, BC is an example of Distributed Ledger Technologies (DLTs) that provides an append-only file across several nodes operating in a Peer-to-Peer (P2P) network structure [15]. When adding a new record into this file, all the participating nodes must agree that the record is valid. This agreement will be achieved by utilizing a Consensus Protocol (Section 2.1.2.2). In BC, records are formed into various transactions that are included in linked blocks. These blocks are linked together by hash of the previous block's data; for example, the link in block n corresponds to the hash of block $n-1$. This process defines the Chain of Blocks (CoBs) illustrated in *Figure 2*. If a block gets tampered with or is injected intentionally, the calculated hash value does not match in CoBs and the network considers it as an invalid block. Moreover, the first block in the chain is known as the Genesis Block (GB), which is the starting point of the ledger at the time of network initialization. More details on BC are provided in Section 2.1.2.

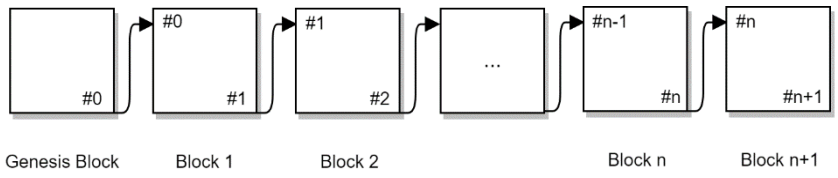


Figure 2 – Simplified version of Chain of Blocks in BC [16]

BC is classified into Permissioned and Permissionless types. Permissioned BCs such as Hyperledger Fabric [17] and R3 Corda [18] provide more trust to the network since they are running with the number of authorized nodes participating in the consensus mechanism unlike the permissionless type [19]. BCs can also operate on an isolated (a.k.a. Private) network instead of

an open (a.k.a. Public) network. Private BCs are controlled by one or more organizations, while Public BCs are open to the public [20]. Although Public-Permissionless BCs such as Bitcoin [13] have desirable characteristics such as transparency and anonymity, they are less secure [15] than Private-Permissioned BCs in terms of network joinability, since every node can join/leave the network freely. However, the main benefit of Public-Permissionless BCs is decentralization; meaning that there is no need to a centralized party to manage the network. It is worth mentioning that Public-Permissionless BCs are vulnerable to the 51% attack where more than 50% of the network nodes (where Proof-of-Work consensus mechanism is used) are malicious and take the control of the network [15].

1.2. Motivation and Problem Statement

The software industry is growing at a very fast pace; therefore, the deployment of a Global Authentication System (GAS) is beneficial and important since it will offer an authentication platform compatible with the developed products and services, irrespective of who the manufacturer is. This will result in better software integration, improved user experience, and will remove the need for physical identifications. On the other hand, e-Services provided by governments in smart cities are migrating to nonphysical online services by employing software solutions, which emphasizes on the government's role as an authority to manage the authentication process. Therefore, GAS is much needed - in both software applications and e-Services - for smart cities governed by governmental organizations. Smartphones (SPs) have also turned to be essential, easily accessible devices that provide the appropriate hardware features (e.g., biometric sensors) needed to support the implementation of GAS. Several authentication systems in this context has been proposed in the literature; their limitations and privacy issues are discussed in Section 2.2. Aside from pointing out these deficiencies, the benefit of our work is that we utilize a

mutual authentication mechanism between two pre-registered parties without the interference of the registration authority. At the end of the mutual authentication, all business transactions between parties are sent to a Blockchain backed storage system that can is accessible at the time of service provisioning. In addition, our system isolates the sensitive computations inside a Subscriber Identification Module (SIM) Card located at the users end. In this thesis, we propose an architecture of a novel authentication system that benefits from a mutual authentication protocol. This protocol is selected based on the criteria introduced in Section 2.2.4 and has been modified according to our needs.

The prerequisite of a Global Authentication System (GAS) starts by developing a lightweight mutual authentication system that can be run in resource-constrained devices with maximum level of security. To realize this, a well-designed registration mechanism is needed. The second requirement of GAS is to keep the user identity safe at the Rest and secure using appropriate cryptographic approaches that require low complexity computations on the restricted devices; this will allow to support two properties, namely anonymity[1] and privacy-preservation. Furthermore, transparency and immutability that a Blockchain network offers at the storage level and the removal of the single authority by using the code-based contracts at the logic side, are two properties appropriate for GAS systems. In addition, information such as personal data should not be exposed to third-party applications that are susceptible to various attacks and privacy issues [21]. To increase the security of the system, we have considered the use of a multi-factor authentication in our work. The mentioned requirements of a GAS system are summarized in *Table 1*. To the best of our knowledge and judging based on what is available in the open literature, the lack of an authentication system mentioned in this chapter with the pointed properties is evident.

[1] More information on "anonymity" is provided in Section 2.2.4.

Hence, we propose a Blockchain-based Global Authentication System (BBGAS) that suggests a secure, scalable, transparent, and privacy-preserving authentication system. This system can support various service provision applications with the scenario of applicability including the replacement of Health Insurance Cards, Bank Cards, Governmental-based services (e.g., Social Insurance Numbers or SINs, passports and citizenship cards, cyber-secure voting services, policing), workforce-related services (e.g., employee management), and private sector companies from small to large workforce size.

Table 1 - Requirements of GAS

Feature	**Identified requirements**
Mutual authentication	Low computation cost, comparable security, low communication overhead
Cryptographic approaches	Low computation cost, low execution time, attack resiliency
Smart Contract based authentication	Transparency, privacy-preservation, removal of the single authority
Authentication factors	Password, Smart Card, Biometrics

1.3. Research Objectives and Contributions

In this dissertation, we introduce a Blockchain-backed authentication system operating on Smartphones (SPs), having the required features to have it deployed and used globally. The main objective of this work is to provide a fast, scalable, secure, privacy-preserving, and transparent system using biometrics in a mobile environment. We satisfy the first objective by utilizing the existing efficient authentication protocols suitable for resource-constrained devices. The scalability of our design work is achieved by distributing the processes between Blockchain (BC)'s

Smart Contract and Smartphones (SPs). The security and privacy of our system is provided by well-suited encryption and decryption techniques, used with the information transported through the communication channels and at the storing ends. Lastly, the transparency is achieved by utilizing a Blockchain backend storage system that eliminates the use of a trusted authority to finalize the transactions and forces the committing rules of transactions. As a result, data are kept safe from tampering and unauthorized modifications at the storage level.

In resource restricted environments such as Smart Cards, the technology selection is challenging. Since one requirement of our work is to select an appropriate mutual authentication between two devices, we first consider several existing authentication protocols proposed for the Internet-of-Things (IoT). These protocols employ cryptographic mechanisms to function vigorously with acceptable level of security. Public key cryptography is the best solution for these environments because of their key size. In this context, Elliptic Curved Cryptography (ECC) provides the same level of security as Rivest-Shamir-Adleman (RSA) does while using considerably smaller key sizes [22]. However, there are countless lightweight ECC-based cryptographic approaches in the literature appropriate for IoT environment that are based on ECC point generators, exclusive-ORs (XORs) and hash functions. Therefore, one contribution in this thesis is the selection of the suitable lightweight mutual authentication technology capable to provide efficient cryptographic functionality when used in restricted devices.

Combining several factors in authentication will increase the overall security of the system since the attacker must have access to all the required factors. Depending on the number of factors included, we might have one, two or three factor authentication schemes [23]. Thus, we are including three factors to the local authentication of the user that consists of Biometrics, Smart Card, and Password in the Smartphone (SP) environment. Password and Biometrics are essential

for authentication of the current user during the system usage, while Smart Card will provide an isolated environment for storing local data and processing the requests.

To support the transparency and data privacy of the users, we propose the usage of the Smart Contracts in the core of our system. Smart Contracts are the best place to reach trust when there is no central authority responsible for making the decision [24]. The main responsibility of these contracts in our proposed system is to decide on the legitimacy of an authentication request and provide the authentication authorization. By this, users can either authorize one party to check their legitimacy or revoke this ability from the party to query the Blockchain (BC) network. Hence, Smart Contract applies the functionality logic of the Blockchain network to handle the authentication requests between Users (URs) and Service Organizations (SOs).

1.4. Thesis Organization

The rest of this document is organized as follows. Chapter 2 reviews the definitions and technologies related to this work. It also provides some related works in this context. Next, the design aspects of the proposed scheme are provided in Chapter 3. Then, Chapter 4 shows the implemented scenario of the proposed system. Afterwards, the evaluation of this system is described in the same chapter. Finally, the last chapter concludes the work and suggests avenues for pursuing related work in the future.

Chapter 2 - Background and Literature Review

This chapter provides a comprehensive study of the technologies used in this work. Section 2.1 categorized the content into five subsections. Firstly, Smart Cards, their types, operating systems, and cryptographic comparison are described. Secondly, the definition of the Blockchain technology and the justification of our Blockchain platform selection are described. Then, the basic of cryptographic algorithms along with their performance analysis are provided. In addition, biometrics mechanisms have been reviewed. Finally, we provided a list of attacks and their counter measurements for both Smart Cards and Blockchain technologies; plus, the software-level attacks for Smart devices are identified. Section 2.2 of this chapter provides a review where Smartphone's Biometric API is described, and various existing authentication frameworks related to our work are described.

2.1. Background

As described in the *System Model* of our design (see Section 3.1.1), this work requires use of some devices and technologies. In this section, we focus on the primary technologies and describe them. We explain Smart Cards, their types and application architectures, followed by the cryptographic algorithms supported by each type. Then, we provide details on Blockchain (BC) technology and explain why we decided to use the Hyperledger fabric, we proceed by providing the preliminaries of cryptographic material that is essential to this work. We follow by analyzing the biometric technologies appropriate for humans and their matching processes. Lastly, we show the security weaknesses of the mentioned technologies and discuss the solutions available in the literature.

2.1.1. Smart Cards

The most significant device in our system is the Smart Card, which is a secure chip physically isolated from the outside world, containing its computation and storage facilities. We consider a card to be Smart when satisfying the following properties: be uniquely identifiable, can perform automated transactions, it is hard to tamper, securely stores data and runs cryptographic algorithms [25]. These cards are divided into six categories:

1. **Magnetic Stripe Cards:** A plastic card with information stored in the magnetic stripe located at the back of the card. Although it was initially widely used for debit/credit cards, it was susceptible to forgery and counterfeiting.
2. **Chip Cards:** A plastic card like magnetic stripe cards with an embedded chip instead of a stripe. They operate in a master-slave mode and obtain their power from the reader that acts as master. Unfortunately, these cards are also vulnerable to attacks, resulting in reading and copying the data, same as magnetic stripe cards.
3. **Microprocessor Chip Cards:** This type satisfies the conditions of the smart cards since they provide storage, have communications capability and contain security features. Microprocessor cards are designed to be tamper-resistant and difficult to copy.
4. **Contactless Smart Cards:** This type of chip card benefits from Radio Frequency Identification (RFID) to provide the card's ID. An example of this type is a key fob utilizing RFID.
5. **The MIFARE Cards:** This is a product of NXP semiconductors [26] to provide contactless mechanism to smart cards. It utilizes a proprietary secret algorithm with a concise secret key (48 bits long) to provide security. Due to the major flaws and the secret key length, the algorithm and thus the keys were discovered.

6. **Smart Tokens:** The Subscriber Identity Module (SIM) could be considered as a smart token that satisfies all the functionality and security aspects of the smart card. It comes in a pluggable format that fits into a contact module.

Beyond the above categories, Smart Cards come with different kinds of Operating Systems (OSs) [25]. OS manages and provides an additional layer of security for applications[2] that reside in the card. At first, smart cards came with monolithic application architecture, having a limited hardware specification. However, with the improvement of Smart Card's hardware, the idea of multi-application deployment evolved. The initiative was to host the OS in the Read-Only Memory (ROM) and deploy applications in Electrically Erasable Programmable Read-Only Memory (EEPROM). *Figure 3* shows the transition from one architecture to the other. *Java Card* [27] and *MULTOS* [28] are the most common OS types for multi-application platforms. Java Card applets run on top of Java Virtual Machine (JVM) that interprets the Java bytecode written in Java programming language for any platform. The most recent version of this OS is V3 which comes in two editions: Java Card Classic and Java Card Connected. The Classic Edition supports resource-restricted devices with limited data types of support. It also comes with an API for biometrics support.

[2] An application that runs in the Smart Cards is called an Applet.

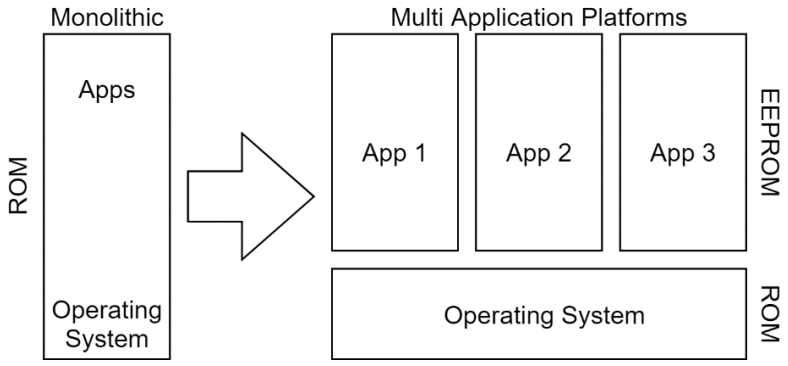


Figure 3 - Monolithic vs. Multi-application architecture in Smart Cards [25]

On the contrary, the Connected Edition introduces a new paradigm into the Java Card world. It provides support for web applications known as Servlets that make these cards act as a powerful network node. In addition, unlike the Classic Edition that employs an Off-card verification approach, this edition provides an On-card Java bytecode verification to confirm the rules of Java Card specification. It also brought an automatic Garbage Collection (GC) to the action instead of the previous best effort mechanism. Finally, in terms of data type support, Connected Edition adds wrapper classes, string manipulation classes, Input/Output classes, and collection classes. Furthermore, Java Card has a specific version of Smart Card for SIM applications called Java SIM, which comes with pre-installed applets and APIs suitable for SIM Cards scenarios.

The MULTOS cards are designed to provide a highly secure and reliable card. Development for this card can be in C, Java, or VB that, with the help of their specific compiler, can be compiled to MULTOS Executable Language (MEL). Then, the compiled code will be run on the card's Application Abstract Machine (AAM). In addition, it supports a rich function library that provides services for cryptography and memory management purposes. The architectures of both cards are depicted in *Figure 4*.

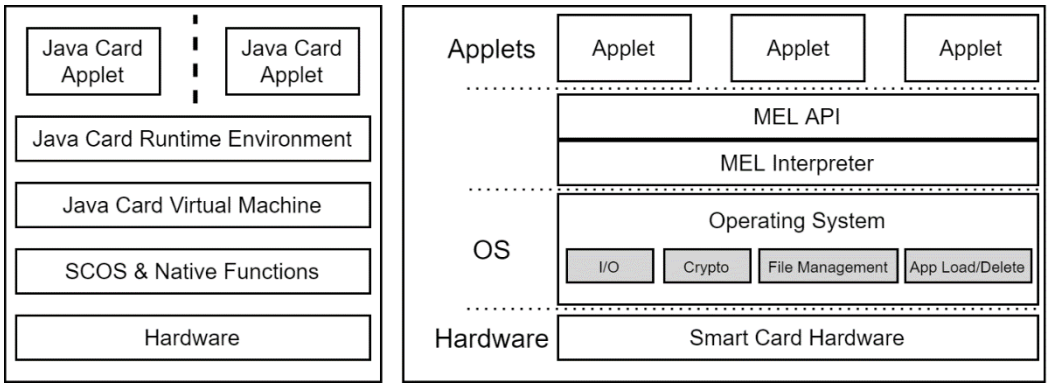


Figure 4 - Java Card (left) and MULTOS (right) architectures [25]

[29] provided a list of available Smart Cards platforms and compared their hardware specifications. According to this comparison, both Java Card and MULTOS support cryptographic functions, as well as contain 16-bit processors with diverse RAM and ROM capacities. Both cards support Symmetric /Asymmetric cryptography with various Key sizes and Hash functions. The specifications in *Table 2* show that either Java Card or MULTOS is suitable for our work.

Table 2- Smart Card Technical Specifications [29]

Platform	**Crypto**	**Type**	**Processor**	**RAM(KB)**	**Storage (KB)** **EEPROM – ROM**
.NET Card	Yes	V2	32-bit (66 MHz)	16	400 - 80
Basic Card	Yes	ZC7.6	-	2.9	EEPROM: 72
Java Card	Yes	V3	16-bit (30 MHz)	6	80 - 264
MULTOS	Yes	V4	16-bit (33 MHz)	2.048	96 - 280
MULTOS	Yes	V4	16-bit (33 MHz)	1.75	18 - 252

One advantage of MULTOS over Java Card is the Modular Arithmetic API (MAA), which supports advanced authentication protocols. These protocols bring security and privacy enhancement for end users. In addition, both cards support algorithms such as DES, TDES, and AES in Symmetric cryptography. Likewise, many algorithms for Asymmetric types such as RSA, DSA, and ECDSA are supported internally. Finally, both cards support the SHA series family in terms of Hash functions, such as SHA-1, SHA-2, and SHA-3. *Table 3* provides more details on the Cryptography specifications of these cards.

In terms of performance evaluation of cryptographic algorithms in Smart Cards, [29] provided a comparison between the aforementioned Smart Cards. The comparison shows that MULTOS has better performance in Symmetric algorithms like AES and Asymmetric algorithms like RSA, while Java Card shows lower processing time in ECDSA with a 256-bit key size for both signing and verifying operations. Overall, Basic Card has the best performance compared to other cards, with some exceptions. For instance, in verification operation for ECDSA and Modular Arithmetic API, Basic Card is not as efficient as some others (e.g., Java Card, MULTOS, and .NET Card). Moreover, the priority goes to MULTOS, Basic Card, .NET Card, and Java Card in hash functions efficiency. *Table 4* summarizes the evaluation between Java Card, MULTOS, .NET Card, and Basic Cards.

Table 3 - Smart Cards Cryptography Support [29]

Platform	Symmetric	Asymmetric	Hash	RNG	MAA
Java Cards	DES, TDES, AES (up to 256b)	RSA (up to 4096b), DSA (up to 1024b), ECDH, ECDSA (up to512b)	MD5, SHA-1, SHA-2, SHA-3	Pseudo RND, TRNG	Not directly supported, exponentiation, ECDH point multiplication
Basic Cards	DES, TDES, AES (up to 256b)	RSA (up to 4096b), ECDH, ECDSA, ECNR (up to 521b)	SHA-1, SHA-2 (up to 512b)	4B RND, TRNG	Supported (up to 16KB)
MULTOS	DES, TDES, AES (up to 256b)	RSA (up to 2048b), ECDH, ECDSA, ECIES (up to 512b)	SHA-1, SHA-2 (up to 256b)	TRNG	Supported (up to 2048b)
.NET Cards	DES, TDES, AES (up to 256b)	RSA (up to 2048b)	MD5, SHA-1, SHA-2 (up to 256b)	Pseudo RNG, TRNG	Not supported

Table 4 - Smart Card's Performance Evaluation [29] (all values are in ms.)

Platforms	.NET Card	Java Card	MultOS Card	Basic Card
Symmetric				
AES- 128-bit key	32	15	13	12
Asymmetric				
RSA - sign - 1024-bit	179	192	69	72
RSA - sign - 2048-bit	625	725	267	397
ECDSA - sign 256-bit	N/A	104	174	58
ECDSA - verify - 256-bit	N/A	112	228	334
Hash functions				
SHA-256 on 128-bit message	13	14	11	13
Random number generator				
Random of 160-bit	33	21	32	12
Modular arithmetic				
Multiplication with 2048-bit	697*	998*	28	19
Exponentiation with 2048-bit	820	478	385	1180

*: Not directly supported. It must use an RSA tunnel.

2.1.2. Blockchain (BC)

During the last decade, a new technology associated with computer science appeared, called Distributed Ledger Technology (DLT) [15]. This technology provides an append-only approach

to the way we store data over a Peer-to-Peer (P2P) network. Imagine a file in which users can only insert a new line while they are not allowed to modify the previously inserted lines, and it is distributed between several nodes. However, when a new line is inserted, the majority of the nodes must agree (depends on the consensus mechanism) to approve that line to avoid tampering; this process is known as the *Consensus* mechanism. Blockchain is considered the most popular DLT since the introduction of Bitcoin in 2008 [13]. In the remaining of this section, we will continue to describe some BC elements and provide samples for it.

2.1.2.1. Blocks

Blockchain (BC) consists of preliminary data formats called *Blocks*. These blocks are linked together by a hash value that is hard to modify. For example, if an attacker changes a block value, the calculated hash value which is based on the block content will not fit in the chain since it will produce a new hash value; therefore, the block will be illegitimate. Each block consists of two major sections: *Header* and *Content*. The former provides a brief description of the block, while the latter carries the block's data. The hash value described above will reside in the block's header and bring consistency to the network. While validating this value to form the Chain of Blocks (CoBs), some validation conflicts might occur that each Blockchain platform has its solutions. For instance, Bitcoin leverages the Longest Chain Rule (LCR) [16] to mitigate such conflicts. The purpose of LCR is to consider the longest chain as the valid chain and ignores the shorter ones. However, in a permissionless network, the adoption of LCR can result in double spending attack which is an attempt by a malicious node to form the longest chain by producing invalid transactions. A countermeasure to this attack is to force a computation mechanism such as PoW described in the following section (2.1.2.2).

2.1.2.2. Consensus Protocols

In distributed computing, where several nodes work together on one or more tasks, some nodes might act maliciously. Thus, a protocol must observe the activities to ensure availability, integrity, and reliability by replicating the data through all nodes and reach an agreement on the order of entries [30]. A *Consensus protocol* could manage this process. Some examples of this protocol are described below.

1. **Proof-of-Work (PoW)** [15]**:** The first and most admired consensus mechanism for DLTs, introduced by Bitcoin. It is based on the Longest Chain Rule principle, and it comes in the form of a mathematical problem. A node must give an answer to that problem on a trial-and-error basis. This process is known as Mining, and the node which has solved the problem is called Miner. The Miner will be incentivized after mining a new block into the chain by devoting its resources to solve that problem. However, this process is computationally and energy consumption wise expensive. Furthermore, we should not underestimate the possibility of 51% attacks (see Section 2.1.5.2) in this mechanism.
2. **Proof-of-Stake (PoS)** [15]**:** This mechanism came as a substitute to the PoW consensus mechanism to address its mentioned deficiencies. In this mechanism, the miners are called validators, and the process of Mining is known as minting or forging. In other words, the network selects validators who must deposit precious assets (e.g., coins) known as stake. This strategy prevents the timing processes and exorbitant resources which PoW wastes. Moreover, if any fraudulent transaction by a validator happens, it will lose all transaction fees and the pledged stake. Ethereum had announced the intention to switch to this algorithm from PoW. However, the feasibility of the Nothing at Stake [31] attack is evident here (see Section 2.1.5.2). Another problem in PoS is when a miner gains the power of the network by having

most of the stakes. When the miner proposes a fork to the network, other miners join that fork in a fear of losing their stakes. This issue is the focus of several research initiatives.

3. **Proof-of-Authority (POA)** [19]**:** This mechanism is used by permissioned or private ledgers in a controllable way. The selected administrators will guarantee the secure and flawless operation of the authorized nodes. It was initially introduced to the Ethereum ecosystem. However, analysis shows that algorithms such as PBFT fit better and outweigh this algorithm.
4. **Proof-of-Elapsed Time (PoET)** [15]**:** Primarily introduced by Intel Corporation in 2016, it uses the Software Guard Extensions (SGX) available in Intel CPUs. It enables the CPU to run an atomic code batch for process executions which is hard to tamper. The scenario is that each peer must wait for a duration of a randomly generated time. When a participating node finishes its waiting period, it will be identified as a validator for the next round. The PoET mechanism is applied to Hyperledger Sawtooth [32] Blockchain platform.
5. **Federated Byzantine Agreement (FBA)** [31]**:** To know what FBA does, we first need to identify fault types in distributed systems. [33] categorizes these faults into two groups. The first category is *Fail-stop* faults related to node failures that prevent them from participating in the Consensus Protocol. For example, hardware or software damages usually cause these kinds of faults that cause the node to stop responding. The second category belongs to *Byzantine Faults (BF)* which are arbitrary, malicious activities first identified as the Byzantine General's Problem [34]. While *Paxos* [35] and *Raft's* [36] algorithms are solutions for the former category, Byzantine Fault Tolerance (BFT) protocols such as Practical Byzantine Fault Tolerance (PBFT) [37] work for the latter group. The definitions of BF and PBFT are described below:

- **Byzantine Fault (BF):** This problem is introduced when various army units are settled around the opponent city, and each unit follows its own general's commands. The generals convey with each other by a messenger. However, when they want to act against the enemy, some generals might betray and try to hinder the faithful generals from obeying the commands. Therefore, there might be a way for them to understand two properties: First, all trustworthy generals will follow the same plan. Second, a minor number of defectors cannot convince other generals to follow their forged plan [34]. *Figure 5* illustrates this problem.

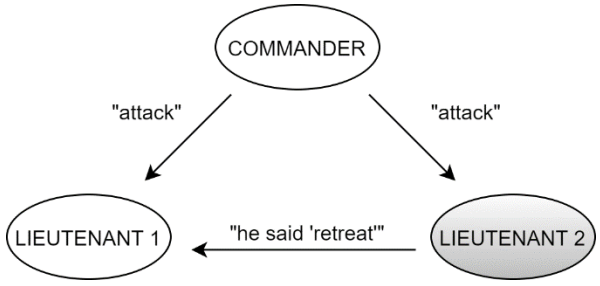


Figure 5 - The Byzantine General Problem [34]

Thus, FBA is a solution to this problem that is introduced in Stellar Consensus Protocol [31] to merge multiple states. Forged states will be omitted by finalizing a single update between participant nodes. Each update *x* is identified by a unique slot number *i,* which is stored in a chronological log. An FBA system has a *verifier* that guarantees all nodes will finally agree on the particular slot's contents, which means that whenever node *v* applies an update in a slot, it is confident that all trusted nodes will agree on it. This action is called *externalization,* and it is defined as "node *v* has externalized an update *x* for slot number *i*" [31]. Besides, other nodes might irreversibly react to this externalization so that it cannot change its decision.

- **Practical Byzantine Fault Tolerance (PBFT)** [37]**:** PBFT provides a state machine through several nodes to tolerate Byzantine Faults in practice. It is worth mentioning that PBFT is the

first operational algorithm in asynchronous systems. This algorithm offers both safety and liveness, when the number of faulty replicas is below the value of $\lfloor (n-1)/3 \rfloor$ where n is the total number of nodes and $\lfloor x \rfloor$ represents the largest integer that is smaller or equal to x. The service is demonstrated as a finite-state automation that is duplicated through several peers in distributed manner. Each replica upholds a service state and performs some service operations. Besides, there are several replicas in each view that are divided into two groups: the core replica is named *Primary*, and the others are called *Backups*. By sending a request from client to the primary replica, the algorithm starts working. Subsequently, the primary replica sends that request to the backup nodes. After receiving that request, backups will immediately run it and transmit a reply towards the client. Afterwards, the client waits to get more replies (f+1, where f is the number of faulty nodes) from other replicas with the equivalent result, which would be the ultimate result. There are three states during the request lifecycle; pre-prepare, prepare, and commit. After completing the operation, replicas post their response back to the intended client individually. Any nodes who prevent to send a reply in the predefined timestamp is known as a faulty replica. *Figure 6* describes this scenario. While C is the Client, Replica-0 is the primary node and backups are listed from 1 to 3. As it shows, Replica-3 is identified as a faulty node [37].

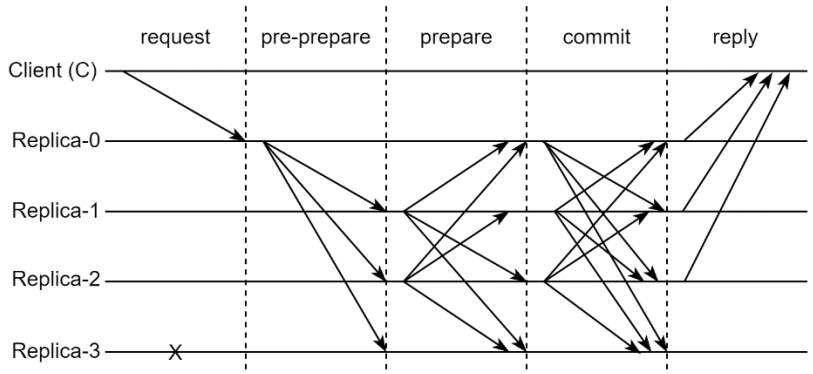


Figure 6 - PBFT algorithm (normal case) [37]

6. **Stellar Consensus Protocol (SCP):** This protocol is based on FBA that is proposed in [31]. Generally, Byzantine Agreement (BA) protocols do not rely on the sensibility of attackers' behavior. Contrasting to prior BA models that membership selection was based on a final agreement, SCP uses an open membership in which the network develops organically. Unlike the PoW and PoS mechanisms, SCP has moderate computations and minimizes the obstacles to new members. The two main sub-protocols in SCP are *Nomination* and *Ballot* protocols. The Nomination protocol generates values for slots which can be used deterministically as a distinct value by nodes. However, it has two limitations. Firstly, peers have no understanding of whether the Nomination protocol has achieved a significant number of agreements. Secondly, even after this, ill-behaved peers might be capable of resetting the nomination process as many times as they want. After the Nomination protocol has been completed, nodes will run a Ballot protocol that utilizes a federated voting system to *Commit* votes (the ballot's value will be externalized for the slot) or to *Abort* them such that the vote becomes unrelated. If a vote gets trapped in a dead-end state, nodes will try again with a higher ballot.

   [31] also pointed to some limitations in this protocol that are listed as follows: SCP only guarantees safety when nodes choose acceptable amount of quorum slices. Whenever user has

option to set the security parameters, the possibility of doing it wrong is inevitable. In addition, safety does not promise other security matters that might occur in federated voting systems. Significantly, fully trusted nodes could force the network to acquire information when it comes to financial systems. Moreover, some nodes on the input side might try to stop transactions by applying a filter to produce the correct output. Finally, when well-behaved peers admit all transactions, the combine function calculates the union of these transactions, and with intact nodes, this filtering eventually will not stop the victim transactions, but might impose delays.

2.1.2.3. Other BC elements

In addition to Consensus protocols, two other BC elements can be listed as *Permission Models* and *Smart Contracts*. [15] divided the Permission Models into Permissionless and Permissioned modes. In the first model, any node can join the network and start mining the blocks; this model is also called Public blockchain such as Bitcoin [13], Ethereum [38], and Cardano [39]. On the contrary, the second model requires registering the participants in the network beforehand, and the activities are audited [30]. Some examples of Private blockchains are Hyperledger Fabric [17], Multichain [40], and Corda [18]. Smart Contracts are programs that run on a ledger to achieve the state by validating the pending transactions. These programs are the best place to provide the logic of the transaction validations.

The role of a Smart Contract and its usage is well-implemented in *Trustee* [41] that offers a secure and low-cost operation in the Vickrey Auction system on top of the Ethereum. Trustee benefits from Intel Software Guard Extensions (SGX)[3] , which uses *Enclaves (E)*. It also offers a verification step with a low-priced fee in comparison to similar protocols. Nonetheless, some

[3] Provides a Trusted Execution Environment (TEE) for Intel architecture.

reports show Intel SGX is vulnerable to multiple side-channel attacks, leading to stealing information in the enclave. Therefore, the Trustee uses a combination of *Intel SGX* and a *Smart Contract (C)* to deal with this issue. *C* in this system provides two benefits: firstly, it freezes the participant's deposits for a while; secondly, it could assess the winner selection since the bidders trust it. In addition, a middleware party called *Relay (R)* provides a communication channel between *C* and *E*. *R* will forward messages from *E* or an auctioneer to the *C*. The Trustee also follows two encryption protocols. First, it uses the Elliptic Curve Digital Signature Algorithm (ECDSA*)* to check the validity of the bid results by signing transactions generated by *E*; Worth noting that key generation happens when creating an Ethereum account in the enclave in the initialization step of the *E*. Then, it utilizes Elliptic Curve Integrated Encryption Scheme (ECIES) to provide a secure communication channel between *E* and *C*.

The auctioning process starts when the auctioneer deploys the *C* on Ethereum and shares the published address with parties who want to participate in an auction. Then, the auctioneer will initialize the *E* via *R*, leading to issuance of E's cryptographic keys. After that, the auctioneer will start the auction on *C* by calling the "StartAuction" method via *R*. Then, bidders can join the auction if they want by submitting their request to *C*. Next, *R* will fetch all the bids from *C* securely and send them to *E*. Finally, the enclave will unseal the bids (decrypt using the shared key) and announce the winner by signing the result and send it to *R*. Finally, *R* forwards the message to *C*, and the process ends with the confirmation of the winner via *C*.

2.1.2.4. Our BC network Selection

The selection of a Blockchain network instead of a centralized database in this work is based on the decision tree provided in [42]. First condition in this tree is that whether there exist multiple writers and since we have three parties in our work, this condition is satisfied. The second condition

that we also face between parties is lack of trust. Next is the existence of a trusted party in the network. The immutability of the transactions as well as the importance of the scalability are considered in this decision tree. Since all the above criteria match our work, we can benefit from the use of a blockchain. Next is to decide on utilizing a public or private blockchain and since we consider verifying transactions within specific parties, we benefit from private blockchain. Furthermore, we consider the use of Permissioned BCs in our work since we have a trusted party, a.k.a. GIDSP (see Chapter 3), that supervises the participant nodes. Therefore, these kinds of BCs provide security and privacy merits compared to Permissionless counterparts because of the authorization process before any activity. They also provide better scalability, and the computation costs of the Permissioned BCs are lower than the Permissionless while they satisfy the transparency[4] property of the BC [43]. However, in Permissioned BCs, we have decided to choose Hyperledger Fabric [17] between various available options such as Quorum[5] and Corda R3 because of the improved throughput, latency, and scalability[6] justified in [44][45]. Furthermore, [46] shows that in a network of 100,000 participants with a total transaction of 200, the latency is a negligible fraction of a second.

2.1.3. Cryptography

Throughout this work, we deal with cryptographic mechanisms in various parts. We start by analyzing the best approaches suitable for BC and then evaluate the best approaches for user's biometrics that reside inside the Smart Card. *Symmetric Cryptography* refers to a way that a single secret key shared between parties is used for encryption and decryption processes, while in

[4] Everyone with authorized access to the network can access the information.
[5] A permissioned implementation version of Ethereum.
[6] Throughput: committed transaction/second, Latency: response time/transaction (in seconds), Scalability: number of serviceable participants in the network

Asymmetric Cryptography, two diverse but related keys (a.k.a. public/private keys) are used by each side [47]. It is worth noting that Symmetric Cryptography must utilize a key exchange mechanism, as well as a key changing mechanism to increase the security. Moreover, *Hashing* is the process of converting plain data into a fixed-size digest without the possibility of recovering the original data (one-way).

A fuzzy logic evaluation in [48] studied the performance of the most famous cryptographic algorithms in the group of *Symmetric*, *Asymmetric* cryptography, and *Hash* functions suitable for BC. The evaluation is weighted based on four factors, shown in *Table 5*. In the Symmetric group, four algorithms, namely SM4, AES, DES, and 3DES, have been evaluated and received the overall scores of 91.33, 95.9, 86.85, and 88.61, respectively. Therefore, the AES algorithm reached the highest score in this set. In the Asymmetric group, for SM2, RSA, and ECDSA algorithms, the total scores of 92.69, 85.87, and 95.74 are calculated, respectively. Thus, the pioneer algorithm for BC in this group is ECDSA. Finally, the most efficient Hash algorithm for BC is SHA256, reaching the score of 92.19 and outperforming SM3, SHA3-256, and MD5.

Table 5 - Evaluation criteria of Cryptographic algorithms in BC [48]

Priority	**Parameter**	**Weights**
1	Security	0.6238
2	Computational efficiency	0.2150
3	Hardware and language support	0.1034
4	Resource consumption	0.0578

The selection between Symmetric and Asymmetric algorithms is also a challenge in our work since it has a direct impact on the processing time and the security of our protocol. According to [47][49],

Symmetric algorithms are faster than Asymmetric ones, and their memory consumption is also lower because of the size of their keys. For example, AES in the Symmetric group uses a key between 128, 192, and 256, while RSA in the Asymmetric group utilizes a key size of 1024. *Table 6* demonstrates a list of cryptographic algorithms alongside their key sizes [47]. While Symmetric algorithms are convenient for bulk data encryption in terms of processing efficiency, Asymmetric algorithms are more secure because of the computational difficulty associated with achieving breaking of public/private keys.

Table 6 - Cryptographic Algorithms Key Size [47]

Algorithm	**Key size**
DES	56
TDES	168
AES	128, 192, 256
RSA	1024
DH	Up to 2048
ECC	112 to 512

Based on [50] and [51], the merits of Elliptic Curved Cryptography (ECC) in asymmetric algorithms outweigh other algorithms such as Rivest-Shamir-Adleman (RSA) and Digital Signature Algorithm (DSA). While the key size of ECC-based algorithms is considerably shorter (160 bits), it provides the same security level as RSA (1024 bits). Therefore, this algorithm is suitable for smart cards and resource constraint devices. By increasing the key size, we could even increase this level of security to a higher degree.

2.1.4. Biometrics

Because of the biometrics importance in our work, we reviewed several biometric mechanisms before making our choice. Since the end users of our system are humans, and we want to include their biometrics characteristics in a SIM card, we have taken into consideration the level of security a certain biometric method for humans offers, jointly with the suitability of executing it in a processing and storage resource-constrained environment, as is the case of executing algorithms in Smartphones and even in (the more challenging environment of) SIM cards.

Generally, biometrics characteristics for humans are divided into *Behavioral* and *Biological* [52]. In behavioral biometrics, user habits are considered the base template for the matching system; examples of these habits are voice, signature, or keystroke. On the contrary, biological biometrics consider the user's physical characteristics such as the face, fingerprint, hand, or Iris. It is worth noting that each type has some benefits and drawbacks. The benefit of the fingerprint is that it provides intermediate security with good computation complexity and high reliability. In comparison, Keystroke-based biometrics are considered weak in security but have lower computational costs [53]. Since in this work the end user's devices are Smartphones, we focus on biological biometrics that are implemented on these devices and can be expanded to behavioral biometrics when Smartphones support them. [53] provides a comparison between various biometrics authentication mechanisms. We compared the usage of biological authentication with the simple password mode in *Table 7* in terms of security, usage simplicity, and technical requirements. It can be concluded that every mechanism has several upsides and downsides, and it is highly dependent on the use case to decide which one would be the best fit.

Table 7 – Biometrics comparison [53]

Biometrics	Usage	Security	Requirement	Description
Password	Simple	Weak	String comparison	Breakable
Fingerprint	Simple	Intermediate	Database	Secure, fast, reliable, needs a device, large or small sized
Iris	Simple	Intermediate	Memory	Accurate, stable, affected by diseases
Face Recognition	Simple	Intermediate	Database and memory	Simple, easy implementation, affected by the effect of hair and light
Palm Vein	Simple	Strong	Memory	It cannot be used on mobile devices
Brain wave	Complex	Strong	Sensor and memory	Not breakable, cannot be used on mobile devices

The process of matching biometrics for authentication purposes can be either online or local. A central database such as a Cloud service is considered in the online mode, while in the local authentication the biometrics samples will be stored in an authenticator device. In both cases, the main concern is to protect the biometrics templates in a way that they cannot be regenerated easily as in passwords; therefore, biometrics data must be under significant protection [52]. Similarly, in the Smart Cards, we divide the storage and matching processes into *Off Smart Card Matching* and

On Smart Card Matching. In the first approach the original biometric template is stored in the Smart Card, and an external device such as a Smartphone conducts the matching process by first fetching the original template from the Smart Card and then reading the real-time[7] template from its sensor. In the second approach, an external device reads the real-time template of the biometrics from its sensor and provides it to the Smart Card. Then, the comparison of these two templates occurs inside the Smart Card that also carries the original template [54]. The latter approach provides better security for the biometric data since the storage of the original sample and matching process between two templates are happening inside the Smart Card that is isolated from the outside world. Thus, the external device acts as a read-only device to provide the real-time template to Smart Card without having access to the original biometric file. *Figure 7* provides a clear view of both approaches.

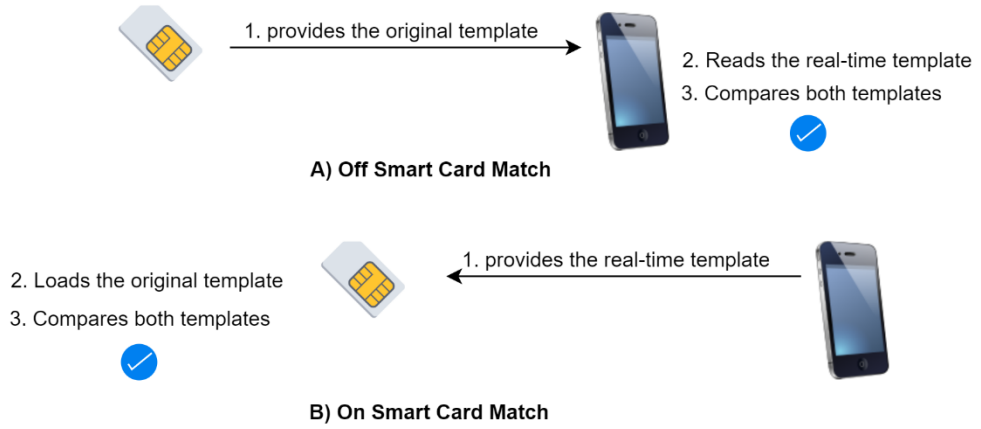


Figure 7 - Biometric matching approaches

[7] Real-time means fetching or reading the current user's biometrics e.g., fingerprint from sensor.

2.1.5. Cybersecurity

In this section, we identify the security risks of the employed technologies in our work. We start by introducing the potential attacks that could be launched against a Smart Card, followed by introducing various attacks to Blockchain. In addition, we provide information on software vulnerabilities and countermeasures.

2.1.5.1. Smart Card Attacks

Smart Cards are a critical component in our work since they store users' biometrics and process the local authentication in Smartphones. Therefore, we need to recognize possible attacks and solutions to offer a secure application and preserve users' privacy. Two major cybersecurity attacks against Smart Cards are known as *Side-Channel* and *Fault Injection* attacks. The main objective of these attacks is to obtain information about cryptographic operations in order to recover the secret keys. In addition to these attacks, we listed five more attacks and provide their description.

- **Side-Channel attack** [55]: In this attack, the attacker observes the device by monitoring some data such as execution time and power usage of the processes. The result of this monitoring is to analyze the obtained values and acquire the main operation. The most significant type of this attack is related to power analysis that detects the device's power consumption to understand the cryptographic operations. Two variants of this type are Simple Power Analysis (SPA) and Differential Power Analysis (DPA). In the first type, one or a small fraction of data is used to measure the power usage, while in the second type, a large scale of data will be analyzed to achieve the usage difference leading to the secret key observation by breaking the

mask[8] of it. A counter measurement to this kind of attacks is to utilize randomness in the cryptographic processes and reduce the signal-to-noise ratio using random delays.

- **Fault Injection attack** [29]**:** In Fault attacks, the attackers will interfere with the cryptography calculations and the running environment. This attack benefits from the existing faults in cryptographic algorithms to recover the secret keys. The most famous attack in this field is Differential Fault Analysis (DFA) that utilizes a fault model. Based on this model, some information will be injected into the device, and the attacker will examine the obtained output data. Therefore, using the faulty or non-faulty results and a nominated key, a calculated differential path will be compared with the expected one. If they do not match, the nominated key will be removed from the candidate's list, and the process continues [55]. Adding techniques such as *Checksums* to verify data integrity, *Variable Redundancy* to reproduce a variable in memory, *Ratification Counters and Baits* to control the number of execution of suspicious code, and the built-in *Firewalls* are considered as precautionary steps to these attacks.
- **Replay attack** [56]**:** This attack occurs when an attacker holds a piece of information for later usage with modification. The utilization of this attack goes up when the volume of communication data in the network increases. The use of a One-Time Password (OTP) or session token is advised to prevent this attack. Another countermeasure is to utilize a *Timestamp* with a predefined threshold (ΔT) at the receiver. At the receiver, any request that exceeds ΔT must be rejected. Hence, the subtraction of the receiver time (T_2) with the sender time (T_1) must be less than or equal to threshold: $T_2 - T_1 \leq \Delta T$.

[8] Mask: the combination of the original data with other values (e.g., timestamp) before encryption.

- **Masquerade attack** [56]**:** This occurs when an attacker utilizes a valid identity to get unauthorized access over valuable information in the communication channel. This information leakage happens because of a weak authorization process. A countermeasure of this attack is to include *Two-step Authentication* processes such as a combination of username and password with a One-time Password (OTP) mechanism.
- **Modification attack** [56]**:** This is when a third party listens to the communication channel and changes the values of the original message between sender and receiver; by doing so, none of the parties will be aware of the alteration of the messages. A solution to this attack is to apply a robust cryptographic algorithm to the messages alongside timestamps.
- **Man-in-the-middle (MITM) attack** [56]**:** In this attack, an adversary intercepts the communication messages and obtains knowledge about the entire communication process. He then utilizes this information to forge all the messages and alter this process. *Mutual authentication* with a complex process is a precautionary step to avoid this attack. A one-way hash function and inclusion of timestamps also add a level of security to the communication channel.
- **Possible Malware attacks** [57]**:** Smart Cards are susceptible to software-level security attacks that identifying and solving them is also beneficial. *Hidden Command* [57] is the first example of such attacks. This attack applies when a card reader sends commands such as SELECT, READ, UPDATE, VERIFY and AUTHENTICATE to the Smart Card when communicating with it. However, some of these commands are used in the Smart Card's initialization process and must be restricted afterwards. Otherwise, they can be misapplied and misused by untrusted applications to retrieve information from the card. Next is *Rogue Applet* [57] that occurs in Smart Cards with the multi-application architecture and endangers the private information of

applets. A solution to this is to utilize the built-in firewall that is operating inside the card to monitor the interactions between applets. Moreover, *Illegal byte codes* [57] can also obtain access to the Smart Card's memory and bypass the firewall. The countermeasure to this attack is to use the card's verifier that verifies the byte code before execution.

2.1.5.2. Blockchain Attacks

Blockchains (BCs) are suffering from various attacks, which are inevitable in a Cyber-world consisting of a huge number of participants. Therefore, a successful consensus protocol must be resistant to them. Some attacks relevant to the consensus mechanism are described below.

- **51% attack** [31]**:** In a BC network where more than half of the nodes (miners) control the power, they can prevent a transaction from being committed, leading to double-spending. At first, attackers are less than 50%, but they incentivize nodes who join them and increase their power. Interestingly, Bitcoin and Proof-of-Work (Pow)-based mechanisms are susceptible to this attack, as mentioned in [15].
- **Nothing at stake** [31]**:** This occurs in the Proof-of-Stake (PoS) consensus or other stake-based mechanisms. In this attack, nodes that already used their collateral will re-spend that money by changing the history from a specific point where they had a significant stake, while they have nothing at their stake. A solution to this issue is to use a combination of Proof-of-Stake (PoS) and Proof-of-Work (PoW) consensus mechanisms to establish an irreversible checkpoint.
- **Sybil attack** [31]**:** In Byzantine Agreement (BA)-based protocols, the possibility of Sybil Attack is unavoidable. In this type, wicked nodes will join the system multiple times and gain control over trusted ones. Therefore, the consensus must avoid malicious nodes from joining the network frequently and exceed the network's failure tolerance.

2.1.5.3. Software Cracking

Software applications that are installed on the Smart devices such as Smartphones also encounter attacks. An example of such attack is Software Cracking [58]. In this attack, attackers use techniques to modify the binary code of the installed application using various tools. It can be divided into Static and Dynamic attacks. In *Static* software cracking, an attacker replaces the code batch with his/her own arbitrary code before executing, while in *Dynamic* tampering, the attacker changes the code that is currently running in the memory. There are several countermeasures to this attack that is listed below and are described in [58]:

1. **Protection guards' network:** This solution benefits from code segments or guards. Guards can be either a Checksum Function or a Repair Function. The checksum identifies the modification by calculating the hash value of the original code and verifies any changes to the code. The repair function acts as a self-heal function that can restore the original code when it detects alteration. However, this approach has some drawbacks: (1) It is easy for the attacker to detect the reading process of the code. (2) The attacker can change the code while it is running by using debuggers. Oblivious hashing is a solution to the mentioned weaknesses.
2. **Oblivious hashing:** This comes to cover the weaknesses of the previous method. It benefits from a hash value computed in a regular function process and not revealed to attackers. The hash value is computed by applying the instructions in the source code or memory.
3. **Remote tamper-resistance:** This works based on a client-server model where a running program on a client must communicate with a trusted program on the server host to be validated. The approach needs a stable connection between client and server operating on the same local area network.

4. **Monitoring running environment:** Another solution to software cracking is to check if the application runs in a trusty environment and ensure the attacker is not changing the execution environment. In hash-based tamper-resistance solutions to deal with software cracking, a code can be accessed in two ways: (1) as instructions or (2) as data. When reading as data, the unmodified version of the code must be used, while in executing as instructions, the modified version is used. These lead to mapping the instructions to one address and the data to another. Because of that, an attacker will be able to copy the main program (P) into P_{origin} and modifies the P as he/she wants. Then, he publishes the modified version into P'. In the end, the attacker modifies the system kernel to read the data from P_{origin} while he sets the instructions to be read from P'. Therefore, the attacker can alter the instructions at his will. The countermeasure to this attack is to add a piece of code to the program so that it checks if the execution environment should be trusted.

Table 8 - Tamper-resistance approaches [58]

	Static tampering	**Dynamic tampering**	**Performance overhead**	**Resistance**
1. Protection guards' network	Yes	No	Average	Bad
2. Oblivious hashing	Yes	Yes	High	Good
3. Remote tamper-resistance	Yes	Yes	High	Bad
4. Monitoring running environment	Yes	Yes	Average	Good

Table 8 provides a comparison between all four tamper-resistance approaches in Software Cracking attack. As it shows, approach number two and four have better resilience to this attack, while the former imposes high performance overhead.

5. **Flushing:** In addition to above mentioned solutions to Software Cracking, applications also utilize caching to avoid regular tasks, for speeding up the operations. This persuades the attacker to analyze the cache memory. The caching is especially prevalent in decryption processes where the plaintext remains in the cache. A solution to this problem is to flush the cache memory in regular intervals to prevent the adversary from accessing the cached information [59].

2.2. Literature Review

One of the features Smartphone vendors offer for biometric-based authentication is the Biometrics API that is suitable for software applications. Therefore, we start by giving a review of existing Smartphone Biometrics APIs and the potential biometrics types that can be supported by Smartphone manufacturers in future. Next, a pilot authentication framework for software applications using the mentioned API is reviewed. Then, a list of authentication solutions is given that has been proposed in the literature to suit the Electronic Governments authentication mechanisms. In each section, we also provide deficiencies and drawbacks of each solution. Finally, we present various lightweight authentication protocols in the literature, appropriate for resource-constrained devices with a comparison table that leads to the selection of the most proper authentication protocol for our work.

2.2.1. Smartphone's Biometrics API

Smartphones have undergone rapid development during the past decade. The manufacturers not only improve the hardware specifications but also introduce innovations to enhance the user experience. An example of such improvement is a biometric-based local authentication system that paces the authentication process for users [9]. By enabling this feature on the Smartphone, a user registers his biometric of choice that the phone supports and utilizes it frequently (e.g., hundred times per day [9]) to benefit from a more convenient authentication mechanism instead of the password-based systems.

Two well-known manufacturers in the Smartphone industry are Apple and Samsung (see Section 1.1.4). The well-known biometrics systems in Apple devices are: "Touch ID", which employs users' fingerprints, and "Face ID", which benefits from face recognition technology to authenticate the user [9]. Similarly, Samsung devices utilize such methods to speed up the authentication processes, which are called by their traditional name "Fingerprint" and "Face Recognition" in Samsung Smartphones. Although both support more biometrics in addition to the mentioned schemes, the future technology of biometric-based authentication is evolving. Behavioral characteristics of the users will be the next generation of biometric authentications supported by Smartphones. Examples of behavior characteristics are the user's voice, gait, finger movements, and touch habits [60].

Both Apple and Samsung provide biometrics authentication APIs for desktop applications running on their OSs. Therefore, the user uses his biometrics to authenticate himself for the application instead of entering his registered username and password for that specific application. This approach speeds up the authentication process and increases user satisfaction. The APIs in these devices work without exposing any biometrics data to the applications, and the biometric matching

process happens in an isolated environment known as TEE (see Chapter 3), either physically or virtually [9][8]. However, none of these systems distinguish between different users' biometrics for authentication. For instance, consider User(A) registers his fingerprint to his Smartphone. If User(B) does the same process and registers her fingerprint to the phone, both users can unlock all the applications on the phone. This issue is a considerable problem for applications such as our work that requires a separation between multiple registered biometrics.

2.2.2. Digital Identity Service

At the time of writing this manuscript, a pilot project called Digital ID is in progress; it is a collaborative effort between Samsung and Mastercard [61]. In this system, a technology like the previously mentioned technology (see Section 2.2.1) is used; it has the ability to share the user's information with a third-party application by user consent. Four roles are involved in Digital ID and are provided below:

1. **Identity Verification Provider (IVP):** IVP is responsible for verifying the Relying Parties (RPs). For instance, Master Card in this project acts as an IVP to manage RPs.
2. **Relying Party (RP):** This party relies on users' information and provides services to end-users.
3. **Trust Provider (TP):** TP is accountable for the biometrics authentication of users in Smartphones. We consider Samsung in this project as a TP.
4. **Users:** Like every service provider application, users are the service consumers of the system.

Figure 8 illustrates the communication between these roles. By utilizing this project in service applications, users have an option at the applications' login page to trigger the Digital ID for their

authentication. This process utilizes Smartphone biometrics API to authenticate the user locally and securely without exposing the user's biometrics.

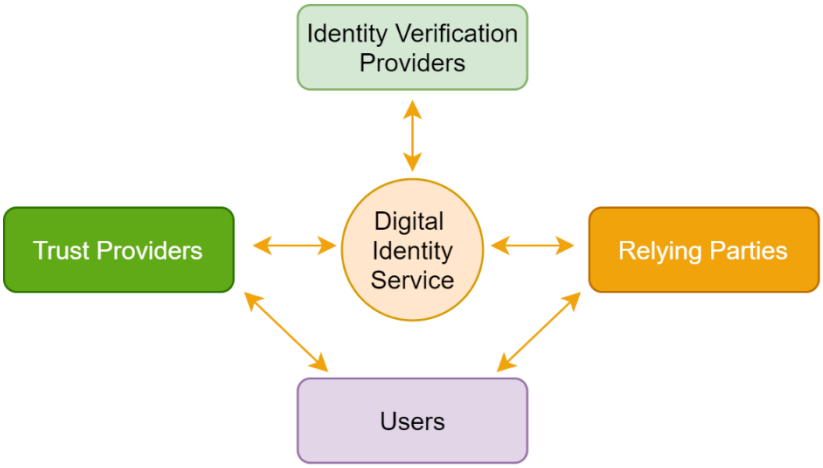


Figure 8 - Digital ID roles [61]

However, this project suffers from the multiple user registrations issue mentioned in section 2.2.1 which results in identity mixture due to the usage of Smartphone's Biometrics API. Take a family, for instance, where it is impossible to provide a Smartphone for children; there should be a possibility of using a single phone for more than one person. Therefore, a better solution would be to register different identities based on their Biometrics and distinguish between them when an individual requires to use his identity. In addition to this, the user's privacy after granting access to an application is not fulfilled in Digital ID since the user cannot revoke the previously granted access from the application.

The comparison of the Smartphone based systems is Sections 2.2.1 and 2.2.2 with our work is shown in *Table 9*.

Table 9 - Comparison of Smartphone's API based Systems

Work	**Support of multiple users on a single Smartphone**	**Users' privacy (data exposure)**	**Support of access revocation**
Smartphones' Biometrics API [9]	×	✓	×
Digital Identity Service [61]	×	×	×
BBGAS (this work)	✓	✓	✓

2.2.3. Electronic Government Systems

The transition to smart cities has brought attention to the requirement of an Electronic Government (E-Gov). This evolution shifts the public services from traditional governments to the electronic version by making use of the Internet. E-Gov services are identified as services that a government offers to Citizens, Businesses, and other governments. For example, issuing driver's license is a service that a government provides to its Citizens. Similarly, a government issues a business permit for Businesses, and it exchanges data internationally with other governments. Although several authentication systems are proposed to fulfill the governmental services, each has its drawbacks. We list some of the systems and point out their deficiencies.

Yang et al. [6] provides a list of technologies that lead to secure E-Gov systems. The first technology that is mentioned in this list is blockchain. It also offers the use of biometrics in smart governance to increase the security of such systems, as well as the comfort of Smartphone usage by users. Moreover, they provide a privacy-preserving framework to be used in E-Gov systems.

Although they benefit from Ethereum Blockchain in their framework, they have not implemented biometric authentication in their work.

AlRousan et al. [62] proposed an authentication system for E-Gov services that benefits from a two-factor authentication scheme. This scheme consists of a combination of biometric and One-Time Password (OTP) authentication. The OTP is generated in the application hosted on the user's Smartphone and the biometric file such as the user's facial image will be sent to the governmental zone for verification. However, this model does not run in an isolated environment (e.g., TEE) and the idea of sending the biometric file to the governmental server over the Internet is susceptible to various attacks. Moreover, the verification process of the biometric on the government zone results in a high processing load on the governmental sites and leads to the single point of failure problem. In addition, this model has yet to be implemented and requires further analysis.

The proposed digital identification system for E-Gov services by Geteloma et al. [63] introduces a web-based authentication system for accessing governmental services in Nigeria. It also emphasizes utilizing multiple factors in authentication mechanisms such as user knowledge, user possession, and user inherence to increase security. This system consists of three layers: Application Layer, Identity Layer, and Authentication Layer. The first layer is where end-user applications reside; examples of such applications are e-Voting, Driver's License, e-Passport, e-Health, and e-Payment. The second layer is a physical Smart Card that hosts cryptographic material for encryptions and signatures. It also encapsulates the functions for identity and session management (e.g., initialization, termination, etc.). Finally, the third layer provides the core authentication functionalities to be used by the Identity Layer. Although this work reduces multiple physical cards into a single Smart Card, the details on biometric matching and Smart Card authentication are ambiguous. Furthermore, this work aims to store the user's data in a centralized

database which does not satisfy the transparency property of the governmental systems as well as forms a single point of failure.

In [64], Maazouz et al. proposed a Mobile ID identification system that eliminates the use of multiple plastic cards. The purpose of this work is to identify citizens when accessing banking services and governmental services with the help of a SIM Card that hosts the certificates and the required application for signing and authentication of the user. This application is inspired by Electronic Identity Cards (e-IDs) used by several countries like Spain [65] as national identification card. An e-ID carries an encrypted version of the user's biometric file (utilizing PKI) that is used to authenticate the user while using the electronic services. For this purpose, this card must be inserted into a card reader alongside a Card Access Device (CAD) that is essential to biometric authentication. However, the authentication process of e-IDs occurs in a remote authentication server that requires exchanging messages from the kiosk to the server. Moreover, the Mobile ID first reads the data inside the SIM Card of the user for online identification and uses some visual information on the User Interface (UI) of the application for the physical identification. Then, it electronically authenticates the user by utilizing a digital certificate residing in the SIM Card via the NFC reader and signs the process which requires a PIN number to be entered into the system. While the author mentioned the usage of the biometric authentication in the proposed scheme, the steps are not described in this paper. Furthermore, this approach utilizes a client-server model like other mentioned schemes.

Table 10 compares all the Electronic Government systems provided in this section.

Table 10 – Comparison of Electronic Government Systems

Work	**Blockchain**	**Support of SOs**	**Biometrics**	**Factors**	**Observations**
Yang et al. [6]	✓	×	×	×	Consumes Ethers
AlRousan et al. [62]	× All the process on government servers	×	✓	Biometrics + OTP	Biometrics file is sent over the Internet, runs on a mobile desktop app
Geteloma et al. [63]	×	×	✓	Smart Card + Biometrics + OTP	Biometrics matching details is not provided
Maazouz et al. [64]	×	×	✓	Biometrics + PIN	Requires additional device, occurs on a remote authentication server
BBGAS (this work)	✓	✓	✓	Password + Biometrics + SIM Card	Benefits from a Mutual Authentication protocol

2.2.4. Lightweight Authentication Protocols

Throughout this work, the authentication of two communicating devices is mandatory. For instance, one communication model in our design is User-to-User (U2U) as described in section 3.1.3. For users to be able to securely authenticate each other using their Smartphones (SPs), an ISIMC must be initiated and hosted in an SP (Section 3.2.1 provides more details on the initialization process). Since the authentication process between two ISIMCs requires to satisfy the criteria of our work, we first define these conditions, then introduce some lightweight authentication schemes in the literature and compare them to choose the best match for our protocol.

The criteria that we identified towards the requirements of the appropriate authentication protocol is listed below:

1. **Security properties:** According to [66], every authentication protocol must meet properties such as User Anonymity, Un-traceability, and Mutual Authentication. In addition, it should be resilient to various attacks. Each property is described below:
    A. **Anonymity:** This defines adherence to users' privacy in exchanging messages to protect users' data such as real identities from attackers [66].
    B. **Un-traceability:** This is coherent to anonymity in a way that users' information in exchanging messages should not lead to tracking users' behaviors [66].
    C. **Mutual Authentication:** An authentication scheme must provide a mutually agreed protocol between two devices with or without the presence of the registration authority [66].

D. **Replay Attack:** When an adversary resends messages to parties involved in the authentication process. Utilizing timestamps and random numbers mitigate this attack as mentioned in [67].

E. **Insider Attack:** When registering parties with Registration Authority (RA), no identities or secrets must be passed through a public channel. This prevents an insider from obtaining secret information (e.g., passwords) related to users [67]. In addition, using arbitrary numbers or nonces by users can also prevent this attack [66].

F. **Man-in-the-middle (MITM) Attack:** This occurs when a third party interferes in the communication between two parties. By this, it can modify the communicating messages and forward them to the destination. Mutual Authentication is a prevention mechanism for this attack [68].

G. **Stolen Smart Card/Device Attack:** It is the case when an adversary obtains physical access to a Smart Card or Device, he/she can extract the users' credentials stored in the device [67]. There are several solutions to mitigate this attack; [68] utilizes randomization, [66] offers the use of a one-way hash function, and [9] wipes the stolen device remotely.

2. **Intermediary device:** Occasionally, an additional device (e.g., Gateway) interoperates between two devices in an IoT environment due to the lack of resources in IoT devices [69]. For instance, it participates in the authentication processes between devices and connects them to the Internet. However, in our work, we do not need such device. Instead, we benefit from lightweight protocols that entail low processing loads on resource-constrained devices.
3. **Cryptography schemes:** In this criterion, we consider the most suitable approaches for lightweight authentication protocols. Since we are targeting resource-restricted devices,

utilization of one-way Hash functions, exclusive-or (XOR) operations, and Elliptic Curved Cryptography (ECC) point addition/multiplication would be the best fit [70][67].

4. **Authentication factors:** The number of factors included in the authentication mechanism will increase the system's security [62]. Hence, the combination of two or more factors leads to a robust system. Examples of such factors are passwords, Smart Cards, and users' biometrics.
5. **Communication and Computational costs:** Since we utilize a SIM card in our work and due to the restricted resources available in IoT devices, the use of a low computational processing/storage is essential [71]. Moreover, [66] expresses that an efficient protocol proposes lower communication and computational overheads. Nevertheless, satisfying the security parameters of a protocol adds extra complexities to this property [68].

Vaidya et al. [72] proposed a two-factor authentication protocol suitable for wireless sensor networks. The scheme benefits from a combination of a Smart Card and a password to authenticate the user's device with a sensor node. It also involves a Gateway node in various system phases such as registration, login, and authentication of devices. Despite the security merits it added to the mutual authentication of the sensor nodes, it partially satisfies our security requirements.

Iqbal et al. [70] introduced an anonymous authentication protocol for Smart Homes that utilizes a controller between two devices. Like Vaidya's scheme, this scheme utilizes a combination of the user's device and password in the authentication phase. However, Yu et al. [73] claim that this scheme has some security vulnerabilities. First, it does not securely store the user credentials in the mobile device. Second, mobile devices in this work are susceptible to physical capture attacks since the work does not implement unclonable-based functions that can prevent such attacks. Third, the proposed scheme utilizes a long-term secret key that is not changed periodically. Lastly,

this scheme is susceptible to the Man-in-the-middle attack since messages are not securely encrypted.

The protocol proposed by Kumar et al. [74] benefits from RFID tags that require radio frequency readers to communicate with the Cloud server for authentication. The proposed mechanism is based on ECC cryptography with efficient computational overhead. However, like other mentioned protocols, it requires an additional party for authentication between two parties. It also does not consider the stolen tag attack in its design. Similarly, protocols in Dey et al. [69] and Challa et al. [75] require an intermediary device to fulfill the authentication processes. The Dey et al. protocol employs Hash functions, exponential multiplication, modulo operations, and Hashed-based Message Authentication Codes (HMACs), while Challa et al. is based on ECC, Hash functions, and XORs. Although Challa et al. satisfies all the security parameters of our work, Dey et al. does not provide any information on the anonymity and un-traceability properties of the work. Moreover, Challa et al. utilizes a combination of three authentication factors in its protocol, namely Smart Card, password, and biometrics, that highly matches our work. Considering all the advantages this protocol provides, its communicational and computational overheads are high compared to other protocols. In addition, the intermediary device for authentication does not fit in our work.

Alzahrani et al. [76] proposed an authentication protocol for a medical system that avoids counterfeiting medications during production and distribution. The system utilizes Near-field Communication (NFC) tags attached to the medicines that carry encrypted information about them. This information is generated at the manufacturer via a Fake Prevention System (FAPS) that can be tracked during the distribution lifecycle. The purchaser then will check the product's originality with the help of a mobile phone and the FAPS. The authentication mechanism between mobile

phone and FAPS server benefits from ECC, Hash functions, and XOR operations. Although the efficiency of the protocol is comparable with other protocols, there is no concrete information regarding its performance in terms of anonymity and un-traceability. Moreover, the vulnerability to insider attacks and stolen tags should also be examined. In our future work, we will provide a survey paper to test all the mentioned protocols and evaluate them under same conditions.

Xie et al. [77] provided an Authentication Key Exchange scheme suitable for IoT devices based on dynamic IDs. Their scheme satisfies the anonymity and un-traceability features that include factors such as Smart Card and password in the authentication phase. It is also an ECC-based protocol with a combination of hash and XOR operations. Despite the low communication costs and message numbers, it provides competitive communication costs as compared to other protocols. However, in the security analysis of the work, [77] does not provide information on attack models such as Insider, MITM, and Replay.

Vangala et al. in [67] introduced an efficient authentication scheme for a Smart Farming platform. The proposed mechanism supports Device-to-Device and Device-to-Gateway communication. It also benefits from a Blockchain backbone to validate the transactions that are formed partially on the edge layer and fully on the cloud servers by completing the transactions' data. The authentication mechanism in this paper satisfies the security parameters of our work. However, the communication and computational overheads are comparable compared to the two following works.

Ying et al. [66] introduced a lightweight approach for 5G networks based on a self-certified public-key scheme. Self-signed certificates have low computation and storage impact in public-key cryptography [78]. The proposed scheme is based on ECC cryptography with multiple hash

functions and XOR operations suitable for resource-limited devices. It also considered a combination of a Smart Card and Password as two factors in the authentication process of the user.

Although authors in [66] provided security analysis in their work and justified our security requirements, Haq et al. [68] claimed that the Ying et al.'s scheme is vulnerable to various attacks such as user impersonation, offline identity and password guessing. Moreover, it is claimed that Ying's work does not satisfy un-traceability and is vulnerable to Stolen Smart Card attacks. It is also noted that this scheme is not a truly two-factor protocol due to unprotected storage of the long-term secret in the smart card. Hence, Haq et al. provided a modified version of Ying et al. to overcome these vulnerabilities. The new scheme is resilient to all the considered security threats concerning our work and imposes only slightly higher computation and communication overheads on the involved devices than Ying et al.'s protocol. For example, Ying et al. requires sending two messages with the size of 1,632 bytes, whereas Haq et al.'s scheme compels two messages with a total size of 2,368 bytes. Furthermore, the imposed computational costs in Ying et al. for both user and server are 1.77 ms., while Haq et al.'s scheme is moderately higher with the values of 3.244 and 3.231 ms. for user and server, respectively. The additional cost of Haq et al. is acceptable since it provides competitive security. The tables below show the comparison between all mentioned protocols in this section. Based on this evaluation, Haq et al.'s protocol matches our criteria.

Table 11 - Security of protocols

Work	Security properties						
	Anonymity	Un-traceability	Replay Attack	Insider Attack	MITM Attack	Mutual Auth.	Stolen Card / Device Attack
Ying et al. [66]	✓	× [68]	✓	✓	✓	✓	× [68]
Vangala et al. [67]	✓	✓	✓	✓	✓	✓	✓
Vaidya et al. [72]	–	–	–	✓	–	✓	✓
Haq et al. [68]	✓	✓	✓	✓	✓	✓	✓
Iqbal et al. [70]	✓	–	✓	–	× [73]	× [73]	× [73]
Kumar et al. [74]	✓	–	✓	✓	✓	✓	–
Dey et al. [69]	–	–	✓	–	✓	✓	✓
Challa et al. [75]	✓	✓	✓	✓	✓	✓	✓
Alzahrani et al. [76]	–	–	✓	–	✓	✓	–
Xie et al. [77]	✓	✓	–	–	–	✓	✓

(✓): satisfies this property, (×): does not satisfy this property, (–): Not Applicable

Table 12 - Communication types and Cryptographic features

Work	**Intermediary device**	**Communication type(s)**	**Cryptography**
Ying et al. [66]	No	**U2SRV**	Hash + ECC (+/.) +XOR
Vangala et al. [67]	No	**D2D**, **D2GWN**	Hash + ECC (+/.) + XOR
Vaidya et al. [72]	Yes	**U2GWN2SN**	Hash + XOR + GWN's Secret key
Haq et al. [68]	No	**U2SRV**	Hash + ECC (+/.) + XOR
Iqbal et al. [70]	Yes	**UD2Ctrl2SD**	Hash + XOR + Symmetric Key (between user & controller)
Kumar et al. [74]	Yes	**RFT2RFR2SRV**	Hash + ECC (+/.) + XOR
Dey et al. [69]	Yes	**U2GWN**	Hash + exponential multiplication + mod + encryption/decryption (DH) + HMAC
Challa et al. [75]	Yes	**UD2GWN2SD**	Hash + ECC (+/.) + XOR
Alzahrani et al. [76]	No	**U2SRV**	Hash + ECC (+/.) + XOR
Xie et al. [77]	No	**U2SRV**	Hash + ECC (+/.) + XOR + server's symmetric key

U2SRV: User-to-Server, **D2D:** Device-to-Device, **D2GWN:** Device-to-Gateway, **U2GWN2SN:** User-to-Gateway-to-Sensor, **UD2Ctrl2SD:** User Device-to-Controller-to-Smart Device, **RFT2RFR2SRV:** Radio frequency Tag-to-Radio frequency Reader-to-Server**, U2GWN:** User-to-Gateway, **UD2GWN2SD:** User Device-to-Gateway-to-Smart Device.

(+/.): Point addition/multiplication, **HMAC**: Hash-based Message Authentication Code.

Table 13 – Authentication Factors and Communication overheads

Work	**Authentication factors**	**Communication overhead (# of Auth Messages, Size in bits)**
Ying et al. [66]	Smart Card + Password	U2SRV: (2, 1632)
Vangala et al. [67]	Temp ID & Pseudo ID (based on Real ID) stored in device	D2D: (3, 2272) D2GWN: (3, 2304)
Vaidya et al. [72]	Smart Card + Password	U2GWN2SN: (3, N/A)
Haq et al. [68]	Smart Card + Password	U2SRV: (2, 2368)
Iqbal et al. [70]	Mobile Device + Password	UD2Ctrl2SD: (4, N/A)
Kumar et al. [74]	Tag + Password	RFT2RFR2SRV: (4,1740)
Dey et al. [69]	Token	N/A
Challa et al. [75]	Smart Card + Password + Biometric	UD2GWN2SD: (3, 2528)
Alzahrani et al. [76]	RFID Tag	U2SRV: (2, 960)
Xie et al. [77]	Smart Card + Password	U2SRV: (3, 1376)

(N/A): Not Applicable

Table 14 - Computational overheads

Work	**Computational overhead (ms.)**
Ying et al. [66]	U: 1.7723, SRV: 1.7719
Vangala et al. [67]	D2D: 20.712, D2GWN: 11.933
Vaidya et al. [72]	U: $2t_H + 3t_{XOR}$ GWN: $6t_H + 6\ t_{XOR}$ SN: $3t_H + 2t_{XOR}$
Haq et al. [68]	U: 3.244, SRV: 3.231
Iqbal et al. [70]	UD: 0.022, Ctrl: 0.0736, SD: 0.03
Kumar et al. [74]	RFT2RFR2SRV: 0.0891
Dey et al. [69]	GWN: $2t_{MOD} + 2t_H + t_{MAC} + t_{HMAC} + 2t_{EN} + t_{DR}$ U: $2t_{MOD} + 2t_H + t_{MAC} + t_{EN} + 2t_{DR}$
Challa et al. [75]	UD: 87.1, GWN: 86.78, SD: 69.36
Alzahrani et al. [76]	U2SRV: 0.00007012 $\oplus$ & $\|\|$ are not included
Xie et al. [77]	U2SRV: 16.87

$\oplus$**:** XOR operation, $||$: concatenation, $\mathbf{t_H}$**:** time duration of hash function, $\mathbf{t_{XOR}}$**:** time duration of XOR operation, $\mathbf{t_{MOD}}$**:** time duration of modulo operation, $\mathbf{t_{MAC}}$**:** time duration of Message Authentication Code, $\mathbf{t_{HMAC}}$**:** time duration of Hash-based Message Authentication Code, $\mathbf{t_{EN}}$**:** time duration of encryption, $\mathbf{t_{DR}}$**:** time duration of decryption.

Chapter 3 - Design of Blockchain-based Global Authentication System

In the design of BBGAS, there are some requirements and restrictions that need to be satisfied. To examine these restrictions, we need to identify the working environment of the Smartphones. Generally, Smartphones utilize two environments: one for processing sensitive information, the other for non-sensitive processes. Android and Apple devices have two functional environments called secure and non-secure [8][9]. Whenever a highly sensitive or protected information needs to be processed in the Smartphone, a platform switch must occur to prevent the non-secure environment interfere with the secure one. For example, biometrics authentication on Smartphones must happen in the secure environment, while the Operating System (OS) a.k.a. Rich OS runs in the non-secure environment. Therefore, OS itself is not safe from attacks and unauthorized access from other applications or untrusted codes, despite the sandboxing approach it provides for applications. The secure environment is called Trusted Execution Environment (TEE) and can be either physically or virtually separated from the rest of the system. *Table 15* summarizes some prominent TEE OSs.

Table 15 - TEE Operating Systems (OSs)

Corporation	**Name**	**Isolation**	**Standard**
Google [79]	Trusty	Intel - ARM	Proprietary
Apple [9]	Secure Enclave	Co-processor	Proprietary
Linaro [80]	OP-TEE	ARM TrustZone	Global Platform API
Samsung [81]	TEEgris	ARM TrustZone	Global Platform API

It is worth mentioning that Rich OS provides more features with Rich APIs, while TEE's APIs are limited. Thus, TEE's focus is to provide more security and not flexibility [82].

In Android devices, the Trusty TEE [79] is responsible for computations such as Passcode Verification, Fingerprint Matching, Digital Rights Management, and Key Managements [8]., *Figure 9* displays the processing architecture implemented by Trusty TEE. In this picture several components are shown which are described below [83]:

- **User App:** Applications that are hosted on top of the Rich operating systems (e.g., Android) such as Calendar, Email, Dialing Pad, etc.
- **Hardware Abstraction Layer (HAL):** HAL provides standard interfaces for specific hardware usages such as Camera, Bluetooth, Sensors, etc.
- **Linux Kernel:** The base of Android operating system (Rich OS) is from this distribution.
- **Public/Private Services:** Trusted services run on top of the secure operating such as Mobile Payments, Encryptions, PIN processing, etc. They can be used either publicly or privately.
- **Trusty Kernel:** The secure operating system that hosts trusted (public/private) services. This is based on Little Kernel Project [84].
- **Secure Monitor (SMC):** A Virtual Machine Monitor (VMM) that controls context switching between secure and non-secure worlds. Intel lightweight hypervisor [85] is an example of SMC. The action of scheduling/dispatching of a service and the required resources happen in this hypervisor.
- **Trusty Library (Lib):** A set of libraries that assist the communication with Trusty services.
- **Trusty Driver:** In non-secure OS provides message exchanges with trusted services.

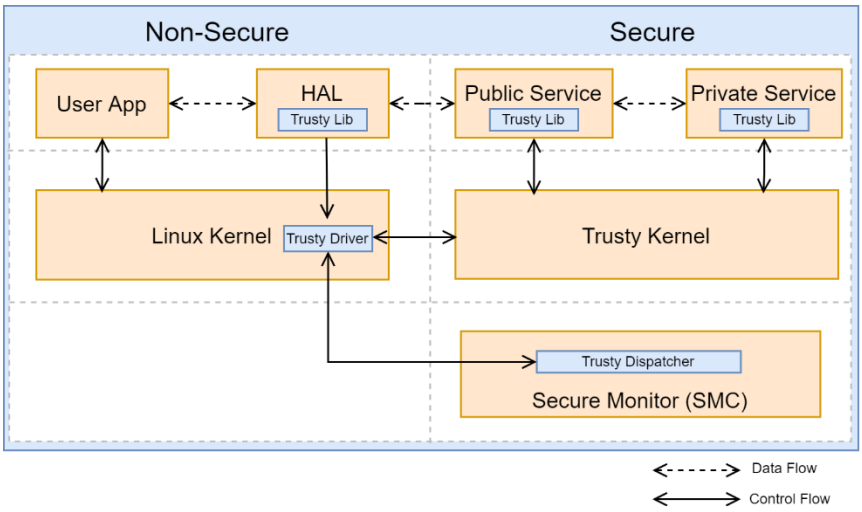


Figure 9 - Trusty TEE architecture in Android [79]

Like Android, Apple devices use a critical component available in most modern devices called Secure Enclave [9] that provides features for encryption and biometrics matching processes, isolated from the main processor. The architecture of this security chip is depicted in *Figure 10*. This picture shows the following items:

- **Secure Enclave Processor:** The secure processor dedicated to computation sensitive information such as biometric authentication, cryptographic operations, etc. It provides its own encryption keys that is generated during manufacturing.
- **AES Crypto Engine:** This engine is dedicated to each apple Smartphone responsible for performing cryptographic operations such as encryption and decryption.
- **Direct Memory Access (DMA):** This is to support read/write operations from/ to the Random-Access-Memory (RAM) without involvement of the main processor.
- **Intel CPU:** This is the main processor of the Smartphone. In the modern Smartphones, apple utilizes its own A-series processors.
- **NAND (Flash) Storage:** A non-volatile memory that stores encrypted data permanently.

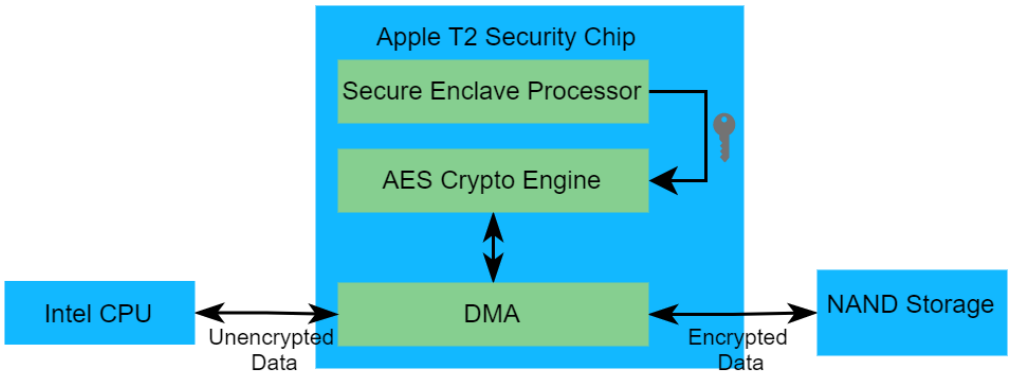


Figure 10 - Apple Security Chip [9]

Both Android and Apple devices use these safe environments for cryptographic calculations and are storing the cryptographic keys in a secure storage called “Secure Storage IC” and “Keystore” respectively in each platform.

In both Android and Apple devices, users can register one or more templates to the phone for biometrics authentication (such as Fingerprint and Face Recognition). All the registered templates are known to the phone as a single user, and it will pass the authentication process when applications request that. Android uses the Hardware Interface Definition Language (HIDL) to connect to vendor library as an intermediatory connection to the device sensors [8]. The vendor-specific library is located inside the TEE and the communication channel to the biometric sensor is dedicated to this library. There are some precautionary steps that a vendor must consider prior to developing its TEE library such as: (1) Encrypting the stored biometric templates with the device-specific key that is assigned to the TEE, and (2) Utilizing SELinux[9] [86] policy that limits the access to the communication channel only to the isolated environment [87]. Moreover, Android utilizes Hash-based Message Authentication Code (HMAC) to enroll and verify biometric and password authentications in TEE [8]. Similarly, Apple provides a serial communication bus

[9] **Security-Enhanced Linux (SELinux):** it enforces the access control, provides protection over data, and guards users from malicious activities [86].

between its sensor and the Secure Enclave to securely transmit the real-time biometrics image [9]. Apple's Touch ID utilizes a shared key between Secure Enclave and the fingerprint sensor which is provisioned at the factory. This session key utilizes AES key wrapping and exchanges using AES-CCM transport encryption. Therefore, the communication between the sensor and TEE is unreadable for the rest of the device with encryption.

To increase the security of our design, we considered the use of a SIM Card (see Section 3.1.1) to host a secure application that is isolated from the rest of the Smartphone. This will increase the security of the system. However, access to the biometric sensors in Smartphones is limited to the TEE environment due to the privacy and security settings of the manufacturers. Thus, we consider two alternative approaches instead of using a SIM Card to overcome this limitation. First approach requires the phone vendors to develop and embed a Trusted Application (TA) to the TEE of the Smartphones, then utilize this TA which has access to the shared encryption key of the sensor for reading the real-time biometrics of the user. It is worth of noting that this TA is equivalent to the embedded applet in our design, which is hosted in the SIM Card responsible for the authentication process. Second is to implement the virtualization architecture defined in [88] by the manufacturers in the Smartphones. In this architecture, three virtualization techniques for TEE are proposed. Two of these virtualization models (C and D in *Figure 11*) can be used in our work to provide secure communication between the sensor and the application that will be hosted in the TEE. The benefit of this approach is that the manufacturer is not responsible for developing the TA; instead, it will provide the foundation of virtualization that makes it feasible to deploy the TAs out of the manufacturer. However, the first approach provides a better security since the TA is controlled by the manufacturers that are considered to be trusted parties. *Figure 11* illustrates the virtualization techniques proposed in [88]. More information on the components of this figure

is provided in the cited article since the detailed description of the architecture is beyond the scope of this document.

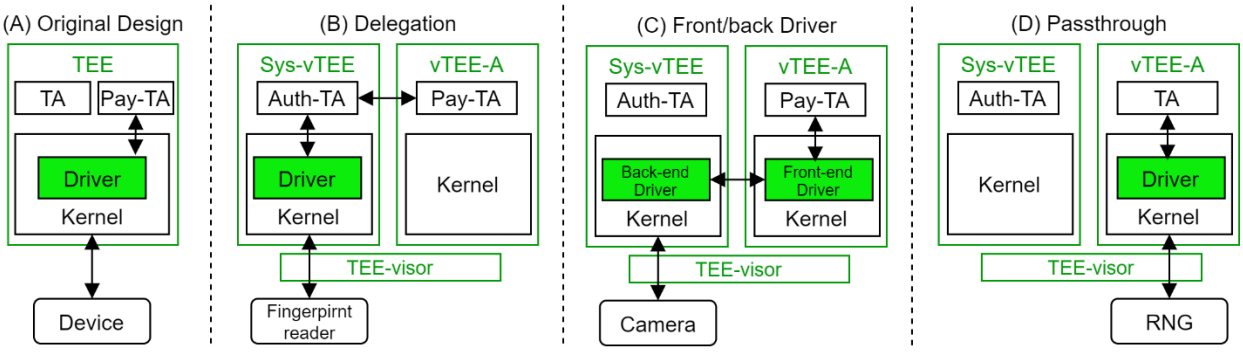


Figure 11 – TEE virtualization device managements [88]

With all the described approaches, we consider the use of a SIM Card in our design to demonstrate the isolated environment for security sensitive computations and disregard the limitation of SIM Cards while introduce two alternative methods to overcome this limitation.

The rest of this chapter is divided as follows: Section 3.1 introduces the system architecture of BBGAS. This section goes to the depth of the device requirements, where we described the mandatory devices in our work followed by the key entities involved in the system. Afterwards, it introduces communication models between system entities. Section 3.2 provides a detailed description of the BBGAS protocol that presents system phases and detailed implementation of each phase.

3.1. System Model

Our system consists of four entities, interoperating with each other. First, we provide a list of required devices for our work. Afterwards, we introduce entities operating in this system, and finally, we describe the interactions between defined entities.

3.1.1. Device Requirements

A total of four device types is mandatory in this work. Their descriptions are provided below:

- **Intelligent Subscriber Interface Module Card (ISIMC):** Refers to a powerful computing chip with cryptographic features that allows us to store securely the user's biometrics inside it. This small device has storage and computing capabilities that can operate solely without any additional tools. However, ISIMC will be inserted into a Smart Device (SD) that provides power for it to be functional and provides biometric sensing device to generate the real-time biometric template for the ISIMC. Examples of SIM chips are regular SIM Cards [25] and Embedded SIM Cards such as Samsung eSIMs with Flash capacities up to 2.5M, with built-in cryptography features [89]. The difference between regular SIMs and eSIMs is that regular SIMs are physical pluggable cards, while eSIMs are mounted on the motherboards. Therefore, the former SIMs can be removed from the phone modules by the users; however, the latter SIMs are embedded inside the Smartphones by the manufacturers, and they can keep several cellular profiles. Furthermore, Java Card [27] and MULTOS [28] are powerful smart card Operating Systems (OSs) that can be used for applet development in our work.
- **Smart Device (SD):** Generally, SD hosts an ISIMC that provides a user-friendly environment for users and sends commands to the ISIMC. It also provides power for ISIMC to make it functional. Any type of SD that satisfies following requirements is appropriate for our design: First, SD must have a SIM or eSIM module to host the ISIMC. Second, it must have a biometric sensor for reading the real-time biometrics templates of the user. However, we consider the use of Smartphones since they are widely available and most of them support the mentioned requirements.

- **Biometric Recording and Storing Devices (BRSD):** These devices will be used to capture biometrics of the user in the Registration Phase of the protocol that are located in the designated physically and cyber secure locations of the service provider (a.k.a. GIDSP). They could also point to the Smartphone's sensors, and ISIMC in the Authentication Phase of our work. Thus, BRSD is a type of device that is responsible for capturing the biometrics and/or storing them. Through our work, we will be storing the biometrics characteristics only inside the ISIMC. We do so to protect the user's privacy.

3.1.2. System Entities

- **Global Identification Service Provider (GIDSP):** A trusted party responsible for registering other entities in the network. It can be governmental institutions such as Government of Canada and Provincial Governments. We provided more details of the registration process in Section 3.2.2. The next task of this entity is to approve Permissioned Nodes (PNs) in the Blockchain network and supervise them to keep the network functional. Worth mentioning that this entity uses a BRSD (see Section 3.1.1) at the registration time of a user to acquire the user's biometrics characteristics.
- **Users (UR):** This entity is the end-user of our system, which utilizes services provided by Service Organizations (SOs). It is also a participant of the Authentication Phase of our work between two entities (e.g., U2U or U2SO, see Section 3.1.3). Thus, it owns a device such as ISIMC which is hosted in a SD.
- **Service Organization (SO):** This entity serves Users (URs) of our system with a type of service that the UR has requested. An example of SO could be a Health Organization (HO) that provides health services to URs. Such services require validating the UR's Health

Insurance Card and inquiring the card issuer to finalize the request and provide that specific service to the UR.

- **Permissioned Node (PN):** We will implement and use a Blockchain (BC) network to deliver a transparent and immutable database for our entities. This network will be operated by some PNs that have been accepted to the network by an authority supervising the network (a.k.a. GIDSP) as in Permissioned blockchains (see Section 1.1). Their number can be scaled up to support more requests and decrease the transactions' confirmation time. More details on our BC network are available in Chapter 4.

The architecture of our system is shown in *Figure 12*. Some required devices are also depicted in this figure to provide a better understanding of the system. As the figure depicts, three primary entities are involved in the system and are interconnected via a Blockchain (BC) network. GIDSP is the principal entity responsible for registering and supervising the network. Note that the communication between GIDSP and UR is during the Registration Phase only and it is offline. Furthermore, each UR owns an SD that hosts an ISIMC with one or more biometric(s) characteristics embedded in it.

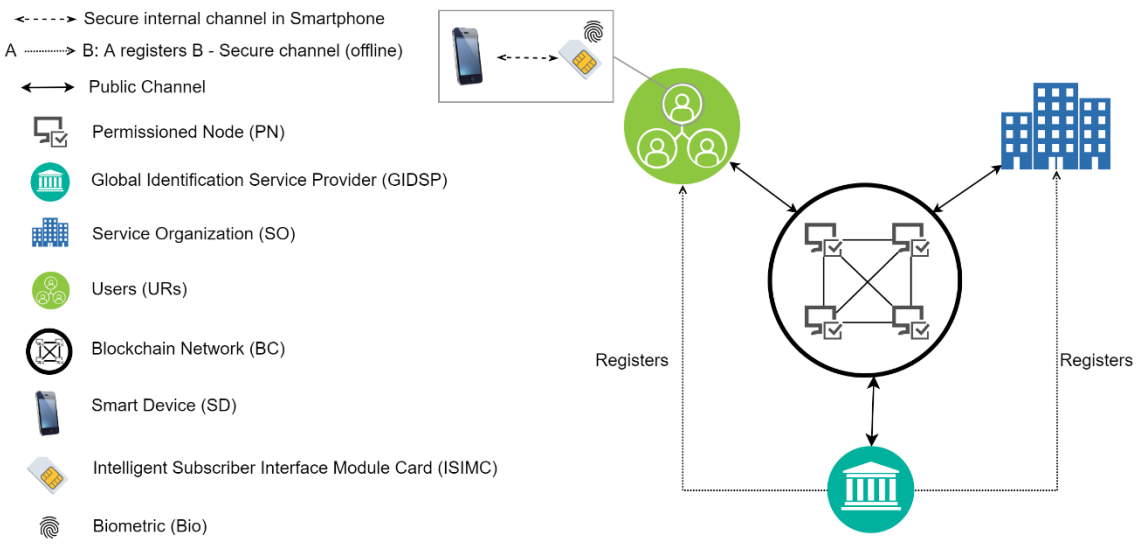


Figure 12 - Architecture of BBGAS

3.1.3. System Interactions

Generally, we have two principal communication processes in our system: *User-to-ServiceOrganization (U2SO)* and *User-to-ServiceProvider (U2SP)*[10]. U2SO is the first communication type in our system that happens between one user and one SO where a service organization provides a service for a user. This model is similar to a communication model that occurs between two users, a.k.a. *User-to-User (U2U)*. Both *U2SO* and *U2U* communication types lead to a service provided by one party and used by the other. As an example, in U2SO communication model, consider a Health Organization (HO) such as a hospital provides medical services to the patients (or users) in the role of a Service Organization (SO). To provide these services using existing systems, the HO first needs to authenticate the user by checking information such as insurance number, identification, etc. However, this process in our system happens on the Blockchain's Smart Contract without revealing data to the SOs that satisfies the privacy-preserving

[10] We assume a User is a human, and it is not software or any type of automation systems.

property of the system with the access revocation capability by the UR. Moreover, maintaining transactions on Blockchain (BC) provides transparency and ensures data authenticity.

Another example of U2SO communication service type is in workplaces. Providing Social Insurance Numbers (SINs) to employers is common in Canada. However, demanding this number by employers in plain text mode can expose it to unauthorized access and fraud. Therefore, utilizing our system will not reveal the actual SIN number but authorizes the organization to consume the individual's SIN numbers on behalf of the user that satisfies the anonymity property. Alternatively, the system exposes the users' *Public IDs*, which are not as confidential as SIN numbers. On the other hand, *U2U* communication model happens between two individuals who wants to share information such as bank account numbers for money transfer. We provide more detail on this communication type in our future work.

The second communications model, *User-to-ServiceProvider (U2SP)*, provides services by Service Provider (SP) to Users (URs) of the system under certain conditions. An example of the U2SP model is when a UR applies for governmental services, such as requesting a SIN number, applying for a passport, and getting a driving license. In addition, this type can be developed to the third type of communication between Service Organization and Service Provider (SO2SP) where a condition must be satisfied. For instance, some services require the candidate to meet specific criteria which SO must check with SP before providing the service, such as age qualification. *Figure 13* illustrates the system's main communication types (in bold) extendable to two more models, represented by numbers 3 and 4.

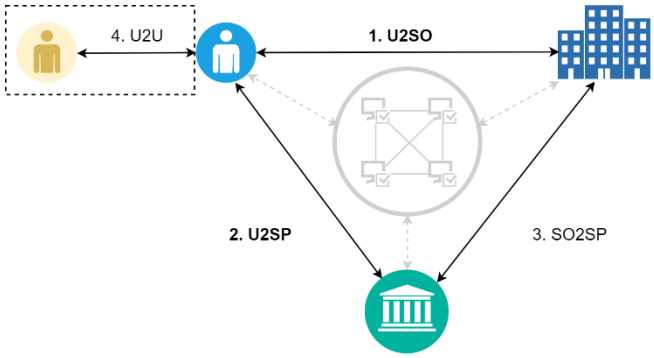


Figure 13 - Communication models of BBGAS

3.2. The BBGAS Protocol

The BBGAS protocol is selected for the reasons provided in Section 2.2. It consists of system phases that are based on the following preliminaries:

- **Hash function:** a function (H) that takes an input as a plain text and returns a fixed length hash value that is infeasible to reproduce the plain text from it. The hash functions used in our design are defined to be noninvertible and collision resistance in [68]:

$$Prob\left[\begin{matrix}(input1, input2) \leftarrow Adv: \\ input1 \neq input2 \; and \; H(input1) = H(input2)\end{matrix}\right] \leq \alpha \qquad (1)$$

with $(0 < \alpha \ll 1)$

According to (1), the probability that an adversary can find the same input for two hash values as well as the same hash output for two different inputs is negligible.

- **Elliptic Curve Cryptography (ECC):** suppose $a, b \in F_p$ be constants where $F_p = \{0, \dots, p-1\}$ and p is a large prime number. A non-regular elliptic curve over the prime finite field (F_p) [90] is defined as below with a point at infinity ($\mathcal{O}$) such that $4a^3 + 27b^2 \neq 0$ holds:

$$y^2 \equiv x^3 + ax + b \ (mod\ p) \quad (2)$$

All the points on the curve $E_p(a, b)$ form a commutative group G that can form a third point after applying the addition rule on two points. Moreover, scalar point multiplication in ECC is defined as multiple addition rules as in $nP = P + P + P$ for $n = 3$ where a point $P = (x_P, y_P)$[11] and $P \in E_p(a, b)$ [75].

According to [68], the authentication protocol chosen in Section 2.2.4 and used in our design is considered secure assuming:

- **Elliptic Curve Discrete Logarithm Problem:** given a point as P over the curve $E_p(a, b)$, it is infeasible to calculate x in $x.P$ where $x \in Z_q^*$ and P generates the whole group with prime order q.
    - Z_q^* is a set consisting of integers $i = 0, \dots, q-1$ for which $\gcd$[12]$(i, q) = 1$ forms a commutative group under multiplication modulo q [91][92]. For example, for $q = 5$ we have a multiplication table for Z_5^* consists of $\{1, 2, 4\}$ as:

Table 16 - Multiplication table for Z_5^*

$\times\ mod\ 5$	**1**	**2**	**4**
1	1	2	4
2	2	4	3
4	4	3	1

[11] x_P is x-coordinate and y_P is y-coordinate of the point P.
[12] GCD (Greatest Common Divisor) is the highest factor that divides each of the integers e.g., $\gcd(6,9) = 3$ [92]

- **Elliptic Curve Diffie-Hellman Problem:** given a point as P over the curve $E_p(a, b)$, it is computationally hard to calculate $xy.P$ in $x.P$ and $y.P$ where $x, y \in Z_q^*$.

Fuzzy extractor: the fixed values in cryptography that are required for cryptographic operations are known as fuzzy values. These values can be extracted from dissimilar but close values. A biometric fuzzy extractor generates a random string k from a given biometric template Bio. Even if the input biometrics are diverse but related, the extractor will extract the same k. Two functions are used in this regard [93]:

- **Probabilistic Generation Function (PGF or Gen):** which is defined as $(k, \tau) = Gen(Bio)$ where Bio is the biometric input, k is the random key generated string where $k \in \{0,1\}^l$ (l is the length of the key in bits), and τ is a helper that will be used in the next function.
- **Deterministic Reproduction Function (DRF or Rep):** which is defined as $k = Rep(Bio^*, \tau)$. This helps to recover the key k by providing a new biometric value Bio^* that is relevant to the previous biometric used in PGF, and the helper value τ generated based on the original biometric in the previous function.

It is worth mentioning that [94] introduced a fuzzy extractor protocol that is considerably more efficient than existing methods. Since we are utilizing ISIMC, fuzzy extractors with high processing speed and less computation (like the mentioned protocol) are convenient for our work. Moreover, an implementation of fuzzy extractor in python is available on [95] which is based on [96].

The terminology used in our system phases is described in *Table 17*. We use these terms in order to describe our protocol.

Table 17 - BBGAS notations

Term	Description
$GIDSP$	Global Identification Service Provider
SO	Service Organization
s_{GIDSP}	Private/Secret key of GIDSP
Pub_{GIDSP}	Public key of GIDSP
SRV_j	j^{th} server of SO
α_{SRV_j}	j^{th} server additional data (e.g., name, type)
ID_{SRV_j}	Identity of j^{th} server of SO
Pub_{SRV_j}	Public key of j^{th} server of SO
UR_i	i^{th} user
ID_{UR_i}	i^{th} user real identity
PID_{UR_i}	i^{th} user public identity
PW_{UR_i}	User's i^{th} password
G	Elliptic curve points group
P	Generator of G
$SK_{i,j}$	Session-key between i and j
ξ_{UR_i}	Encrypted version of questions for the i^{th} user
υ_{UR_i}	Encrypted version of answers for the i^{th} user
$Enroll_{BC}(.)$	Blockchain CA's enrollment function
$Enc(.)$	Encryption function with the provided public key
$\Rightarrow$	Secure channel
$\rightarrow$	Public channel
$\oplus$	XOR operation
$\|$	Concatenation
$H_i(.)$	Hash function ($0 \leq i \leq 6$)
Bio_{UR_i}	User i's biometric
$Gen(.)$	Probabilistic generation used by fuzzy extractor
$Rep(.)$	Deterministic reproduction
k_i	Biometric i's secret key
τ_i	Biometric i's public reproduction parameter
BID_{UR_i,SRV_j}	i^{th} user or j^{th} server blockchain identity
$Data_{UR_i,SRV_j}$	i^{th}/j^{th} user/SOs' registration data

3.2.1. Initialization Phase

This phase is the prerequisite of the subsequent phases of our system responsible for bootstrapping and set default values. The first step is to set protocol parameters such as large prime numbers (p, q) and the non-regular elliptic curve (defined in 3.2) as E: $y^2 = x^3 + ax + b \, mod \, p$. Then, the

Global Identification Service Provider (GIDSP) chooses a private key $s_{GIDSP} \in Z_q^*$ and $Pub_{GIDSP} = s_{GIDSP}.P$ as its public key [66]. Subsequently, seven hash functions will be generated as:

"$H_0: \{0,1\}^* \rightarrow Z_q^*, H_1: \{0,1\}^* \times G \rightarrow Z_q^*, H_2: G \times Z_q^* \rightarrow Z_q^*, H_3: G \rightarrow \{0,1\}^*, H_4: \{0,1\}^* \times G \times G \rightarrow \{0,1\}^*$, $H_5: \{0,1\}^* \times G \times G \times \{0,1\}^* \rightarrow Z_q^*$, $H_6: \{0,1\}^* \rightarrow \{0,1\}^*$" quoted from [68].

The creation of Blockchain identities happens in the *Registration Phase* of the system. Since the Certificate Authority (CA) is located in the Blockchain (BC) network, BC identities will be generated when an entity wishes to register in our system. Moreover, CA uses Public Key Infrastructure (PKI) and Elliptic Curve Digital Signature Algorithm (ECDSA) with key size of 256 to register users in the system. It is worth noting the CA uses ECDSA-with-SHA256 as its signature algorithm [97]. While parties in the BC network such as User (UR) and Service Organization (SO) will be registered during the *Registration Phase*, GIDSP obtains its BC credentials in the *Initialization Phase* (see Section 4.2.4).

3.2.2. Registration Phase

In this phase of our system, the registration processes of entities are described. Two entities are required to be registered and obtain their identities before any type of communication, which are Users (URs) and Service Organizations (SOs). We start with a scenario where a user, known as UR_i, wants to enroll in our system. First, UR_i must be physically present in the GIDSP location and ask for registration. After sending the registration request, the system in GIDSP location goes through the following steps: First, the process starts by checking the user's public identity PID_{UR_i} and the biometric helper parameter τ_i on the Blockchain network. If the user has already been registered, our system refuses to register UR_i; instead, it would ask for other operations, such as

reissuing a stolen ISIMC, which is beyond this phase's scope. Otherwise, the system continues the registration process by requesting further information from the user. *Figure 14* provides the sequence diagram of the registration phase.

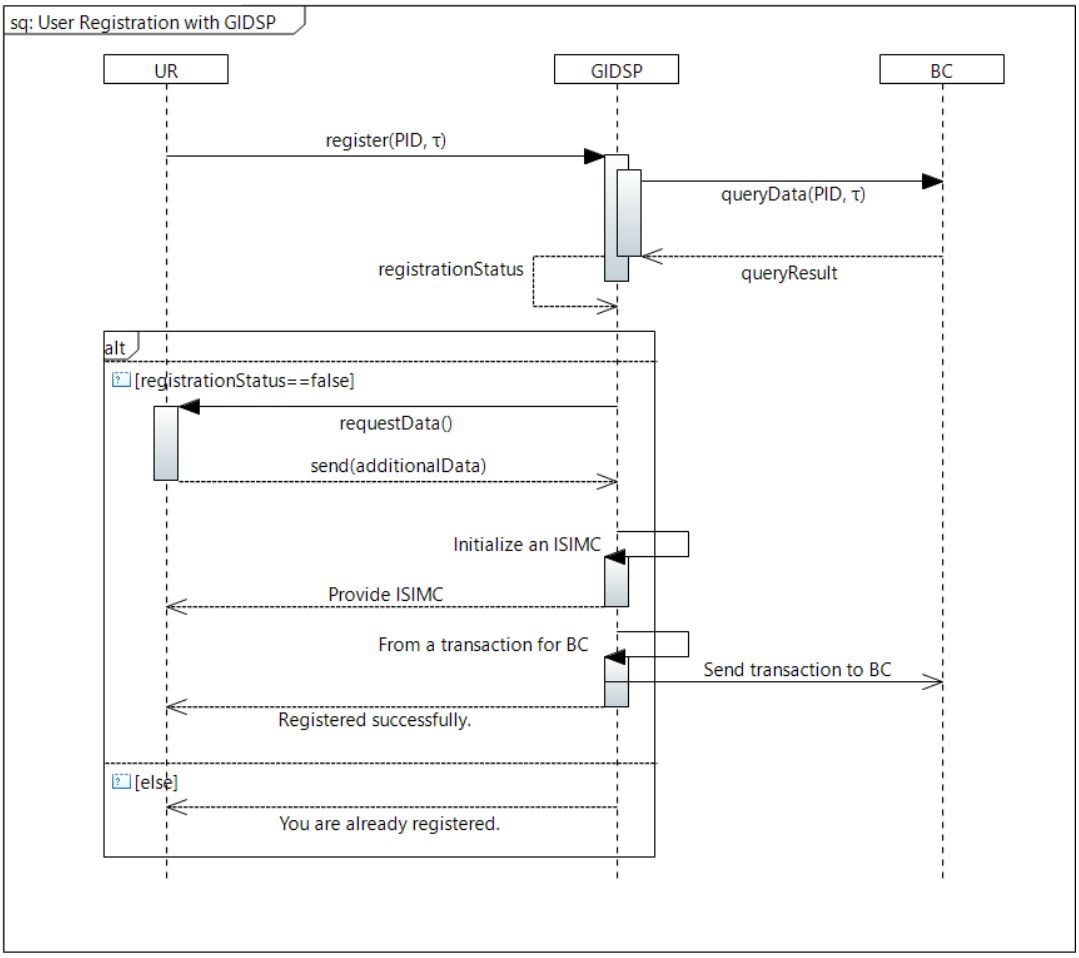


Figure 14 - Sequence Diagram of User Registration with GIDSP

Before continuing with the initialization process of the ISIMC, we need to categorize user's information into two groups. The first group consists of the user's biometric reproduction parameter (described in 3.2), and the hashed version of the calculated user's authentication identification known as $Auth_{UR_i}$. The purpose of $Auth_{UR_i}$ is to aid the local authentication process in ISIMC described in section 3.2.3. The second group of the user's data consists of *Secondary* information such as user identification, and a set of questions generated by the system and

answered by UR_i. The purpose of questions and answers is to help the user in account recovery process that is described in section 3.2.3. Therefore, these questions can be a combination of actual information of the user, such as date of birth, pet's name, and a favorite place. However, it is possible that this information is guessed; a better approach is to generate random strings at the time of registration and include them in the *Primary* Data category that resides in the ISIMC. With this approach, the user's account recovery should include message exchanging mechanisms between ISIMCs and GIDSP nodes that are unreadable by humans, such as complex or encoded strings. Although the latter approach is more secure than the former, we believe the first approach is more user-friendly and convenient for the users. It is worth mentioning that the Primary information will be stored on the user's ISIMC, while the Secondary information will be sent to the BC network. *Table 18* depicts the data, their targeted storage, and their symbols.

Table 18 - Types of users' information

Primary Data (store in ISIMC)		**Secondary Data (store on BC)**	
Description	**Symbol**	**Description**	**Symbol**
Biometrics reproduction parameter	τ_i	**User Public Identification**	PID_{UR_i}
Authentication Identification	$Auth_{UR_i}$	**Questions**	$QSTs_{UR_i}$
		Answers to questions	$ANSs_{UR_i}$
		Biometrics reproduction parameter	τ_i

To continue with the registration process of UR_i in our system, we follow the registration approach proposed in [68]. It starts with the selection of an identity and a password by the user, then UR_i chooses a nonce n_0 and imprints the biometric at the biometric sensing device (or BRSD, defined

in 3.1.1). Next, using the generator function of fuzzy extractor used in [93], the ISIMC extracts the biometric key and reproduction parameter. After that the ISIMC computes $RPW_{UR_i} = H_0(PW_{UR_i} || n_0 || k_i)$ and sends the Registration Request Message $\{ID_{UR_i}, PID_{UR_i}, RPW_{UR_i}, \tau_i\}$ to GIDSP via a secure channel. After receiving the request, the GIDSP chooses a random number $m_i \in Z_q^*$ and computes the $Auth_{UR_i}$. It then replies with Registration Request Reply Message $\{\beta_{UR_i}, Auth_{UR_i}, QSTs_{UR_i}, BID_{UR_i}\}$ to user. At the end of this process, the user stores $\beta_{UR_i}, Auth_{UR_i}, n_0, \tau_i, BID_{UR_i}, Gen(.), Rep(.)$ into the ISIMC (a.k.a. ISIMC initialization). *Table 19* summarizes the user registration processes with GIDSP.

After persisting data in the ISIMC and forming a transaction on the BC network by GIDSP, the system issues an ISIMC to UR_i to be inserted into his personal Smartphone. Since then, whenever UR_i wants to request for a service provided by SOs, UR_i uses the Smartphone that hosts the proposed ISIMC to participate in other phases of the system. Section 3.2.3 provides more details on this process.

The Service Organization (SO) registration starts by sending the Registration Request Message carrying the server identification ID_{SRV_j} and additional information about the SO α_{SRV_j} (e.g., name, type) to the GIDSP over the secure channel. Then, GIDSP chooses a random number $m_j \in Z_q^*$ and computes $Pub_{SRV_j} = m_j.P$, $\sigma_{SRV_j} = s_{SP} m_j^{-1} - H_0(ID_{SRV_j}) \bmod q$. After that, GIDSP replies with $\{Pub_{SRV_j}, \sigma_{SRV_j}, BID_{SRV_j}\}$ message to the SO over the secure channel. After receiving this message, SO stores σ_{SRV_j} locally and publishes its public parameters [68]. *Table 20* summarizes the registration process of the SO.

Table 19 - User registration with GIDSP

UR_i		$GIDSP$
Select $ID_{UR_i}, PW_{UR_i}, n_0$ Imprint Bio_{UR_i} Compute: $(k_i, \tau_i) = Gen(Bio_{UR_i})$ $RPW_{UR_i} = H_0(PW_{UR_i} \| n_0 \| k_i)$ $PID_{UR_i} = H_0(ID_{UR_i} \| n_0 \| k_i)$		
	$\{ID_{UR_i}, PID_{UR_i}, RPW_{UR_i}, \tau_i\}$ $\Rightarrow$	
		Selects $m_i \in Z_q^*$ $\Psi_{UR_i} = m_i . P$ $\varphi_{UR_i} = [H_1(ID_{UR_i} \| \Psi_{UR_i}) s_{GIDSP} + m_i] \, mod \, q$ $\beta_{UR_i} = (\Psi_{UR_i} \| \varphi_{UR_i}) \oplus H_6(ID_{UR_i} \| RPW_{UR_i})$ $Auth_{UR_i} = H_2(\Psi_{UR_i} \| \varphi_{UR_i})$ Generates $QSTs_{UR_i}$ $BID_{UR_i} = Enroll_{BC}(PID_{UR_i})$
	$\{\beta_{UR_i}, Auth_{UR_i}, QSTs_{UR_i}, BID_{UR_i}\}$ $\Leftarrow$	
Stores $\beta_{UR_i}, Auth_{UR_i}, n_0, \tau_i, BID_{UR_i}$ $Gen(.), Rep(.)$ into ISIMC Provides $ANSs_{UR_i}$		
	$\{ANSs_{UR_i}\}$ $\Rightarrow$	
		$\xi_{UR_i} = Enc_{Pub_{GIDSP}}(QSTs_{UR_i})$ $\upsilon_{UR_i} = Enc_{Pub_{GIDSP}}(ANSs_{UR_i})$ $Data_{UR_i} = (PID_{UR_i} \| \xi_{UR_i} \| \upsilon_{UR_i} \| \tau_i)$ Stores $Data_{UR_i}$ on BC

Table 20 - Service Organization Registration with GIDSP

SRV_j		$GIDSP$
Select ID_{SRV_j}	$\{ID_{SRV_j}, \alpha_{SRV_j}\}$ $\Rightarrow$	
		Selects $m_j \in Z_q^*$ $Pub_{SRV_j} = m_j.P$ $\sigma_{SRV_j} = s_{GIDSP} m_j^{-1}$ $-H_0(ID_{SRV_j}) \ mod \ q$ $BID_{SRV_j} = Enroll_{BC}(ID_{SRV_j})$
	$\{Pub_{SRV_j}, \sigma_{SRV_j}, BID_{SRV_j}\}$ $\Leftarrow$	
Stores σ_{SRV_j}, BID_{SRV_j} publishes ID_{SRV_j}, Pub_{SRV_j} publicly		$Data_{SRV_j} = (ID_{SRV_j} \mid\mid \alpha_{SRV_j})$ Stores $Data_{SRV_j}$ on BC

It is worth mentioning that during the registration processes of the UR_i and SO by GIDSP, the BC's identities of both entities will be issued by contacting the BC's Certificate Authority (CA) as described in 4.2.4.

3.2.3. Authentication Phase

This phase comes into action in two scenarios: (1) When a Transaction (Tx) is being sent to the BC network for validating and chaining (2) When a system interaction between parties is occurring in the system.

The first scenario of authentication occurs when any party wants to send a Tx to the BC network. The party can be a UR, a SO, or a GIDSP, defined in 3.1.2. All these parties must be registered with the Certificate Authority (CA) in the BC network before this phase (see Section 3.2.2). Noteworthy, URs and SOs obtain their identities during the *Registration Phase* of the project, while GIDSP obtains his identity during the *Initialization Phase* (Section 3.2.1). Thus, the transactions submitted to the BC will be signed by the parties using their CA's certificates.

In the second scenario, a mutual authentication between two devices is required for further message exchanges between two parties. The process starts when a user UR_i requests a service at a location close to the Service Organization (SO) by clicking on a button on his/her Smartphone to initiate the process. Next, UR_i will be asked for his identity to be verified locally on his/her Smartphone via its ISIMC, based on the information stored inside it (a.k.a. ISIMC Local Authentication). That requires UR_i to input the proper three-factors known as its identity $ID^*_{UR_i}$, biometric Bio^*_i and password $PW^*_{UR_i}$ to the ISIMC. It is important to provide the biometric that was chosen during the registration phase. For example, if the user has chosen the use of a fingerprint as its biometric type, he/she must provide a valid fingerprint to the Smartphone.

The matching process of the biometrics requires the use of the $Rep(.)$ function described in section 3.2. When UR_i provides his biometric to the sensor, this function estimates its biometric key by computing $k^*_i = Rep(Bio^*_i, \tau_i)$ using the value of reproduction parameter stored in the Registration Phase. Then ISIMC extracts $\Psi_{UR_i}, \varphi_{UR_i}$ by calculating $H_6(ID^*_{UR_i} || H_0(PW^*_{UR_i} || n_0 || k^*_i)) \oplus \beta_{UR_i}$. Note that β_{UR_i} was stored in the ISIMC Registration Phase of the user. ISIMC continues by computing $Auth^*_{UR_i} = H_2(\Psi_{UR_i} || \varphi_{UR_i})$ and checks this value with the stored $Auth_{UR_i}$ value in ISIMC. Thus, if $Auth^*_{UR_i} \stackrel{?}{=} Auth_{UR_i}$ does not hold, the ISIMC terminates the process. Otherwise, the mutual authentication process continues. It is worth mentioning that the use of a security threshold can be considered here to increase the security of this process. For instance, a counter will increase by 1 for each unsuccessful local authentication so that if that counter reached the predefined threshold e.g., three, the system would temporarily disable the UR_i, and the user will be challenged by questions that were set during the Registration Phase to reactivate his account. To achieve this, UR_i sends a Reactivation request to GIDSP and

then the challenge process begins. The challenging process is a simple question and answer round in which the GIDSP will choose a random question from previously chosen questions. If UR_i provides correct answers, GIDSP would reactivate his ISIMC; otherwise, the account will be banned, and the UR_i must reactivate his/her account in person. We did not include the reactivation at this stage to maintain simplicity of the work.

The next step in mutual authentication is to establish a Bluetooth connection between the two involved parties after the successful local authentication of the user (a.k.a. biometric authentication). This request will be initiated from SO using Bluetooth API supported in Smartphones and desktop computers. After sending this request to the UR_i's Smartphone, UR_i will either Accept or Reject the pairing request. If he/she accepts, a connection is established between two parties; thus, the communication channel is ready for further message exchanges between them to carry out the mutual authentication processes. Otherwise, UR_i will reject the request, and the process ends. Now that UR_i has accepted the pairing request, the ISIMC can communicate with SO. *Figure 15* illustrates this process.

Now that the Bluetooth connection is established both parties can exchange messages through the Bluetooth channel. Although Bluetooth itself provides some security measurements to make the communication channel safe [98], we add an extra level of security by encrypting the exchanged messages between parties using a mutually agreed session-key.

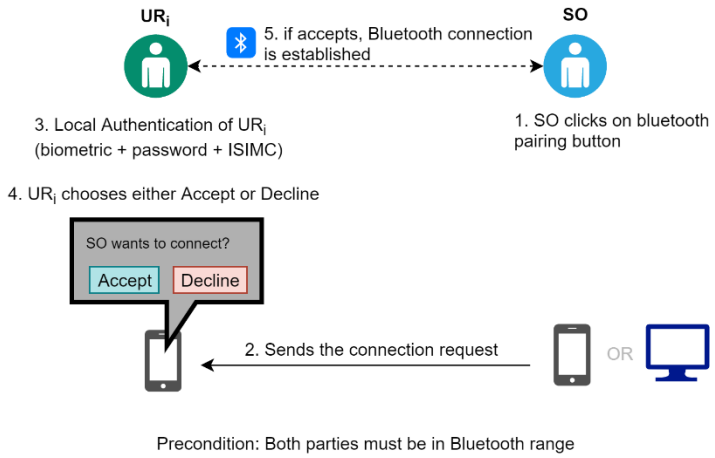


Figure 15 - Bluetooth connection establishment between parties

After local authentication of the user and Bluetooth channel establishment, the ISIMC chooses two random numbers $n_1, n_2 \in Z_q^*$ and computes $K = n_1 . \left[Pub_{GIDSP} - H_0\left(ID_{SRV_j}\right).Pub_{SRV_j}\right], M_{i,j} = n_1 . Pub_{SRV_j}$, $EK_i = n_2 . P$, $\sigma_{UR_i} = \varphi_{UR_i} + n_2 \, mod \, q$, $E_{UR_i} = H_3(K) \oplus (ID_{UR_i} || \Psi_{UR_i})$. It is worth noting that user has obtained ID_{SRV_j}, Pub_{SRV_j} values from the server before computing these values. Moreover, values of parameters such as φ_{UR_i} and Ψ_{UR_i} were calculated by ISIMC prior to the above calculations $(\Psi_{UR_i} || \varphi_{UR_i}) = H_6(ID^*_{UR_i} || H_0(PW^*_{UR_i} || n_0 || k^*_i)) \oplus \beta_{UR_i}$. When done calculating these messages, UR_i sends Authentication Request Message $\{M_{i,j}, EK_i, E_{UR_i}, \sigma_{UR_i}\}$ to the SRV_j over the public channel. It is worth noting that this message is secured by involving the Server's Public Key Pub_{SRV_j} in the calculations. SRV_j , after receiving the Authentication Request Message $\{M_{i,j}, EK_i, E_{UR_i}, \sigma_{UR_i}\}$, it calculates $K^* = \sigma_{SRV_j} . M_{i,j}$[13] and $H_3(K^*) \oplus E_{UR_i} = (ID_{UR_i} || \Psi_{UR_i})$, then verifies $\sigma_{UR_i}.P =$

[13] K was calculated at the User's side, while $K^*$is calculated at the Server's side.

$H_1(ID_{UR_i} \,||\, \Psi_{UR_i})\,.\; Pub_{GIDSP} + EK_i + \Psi_{UR_i}$. If the verification is successful, SRV_j will continue; otherwise, it terminates the session.

Upon completing the verification successfully, SRV_j chooses a random number and calculates $EK_j = m_j\,.P$ and $EK_{i,j} = m_j\,.EK_j$ that leads to the calculation of the session-key $SK_{i,j}$ to be used between the two parties. After calculating the MAC of these values, SRV_j sends to the user over the public channel the Reply Message $\{EK_j, MAC\}$.

At the UR_i side, when the reply message is received, ISIMC calculates $EK_{i,j} = n_2\,.EK_j$ that aids the calculation of session-key in $SK_{i,j} = H_4(ID_{UR_i} \,||\, ID_{SRV_j} \,||EK_{i,j}||\, K)$. Note that K was calculated by the user before sending the Authentication Request Message $\{M_{i,j}, EK_i,\; E_{UR_i}, \sigma_{UR_i}\}$. Then, the user recalculates the MAC^*[14] value and checks whether this value matches the MAC value received from the server, e.g., $MAC^* \stackrel{?}{=} MAC$. If the equality holds, the session-key establishment is successful and both parties have access to the $SK_{i,j}$ for future communications and message encryptions. *Table 21* summaries the steps taken towards session-key establishment.

After establishing a secure connection between two devices, additional messages will be exchanged between the two parties, which are encrypted with the shared session-key. Then, each party will form the relevant transaction that will be sent to the BC for GIDSP.

[14] MAC^* is calculated at the User's side, while MAC was calculated at the Server's side.

Table 21 - Session-key establishment between two parties

UR_i		SRV_j
Obtains $\{ID_{SRV_j}, Pub_{SRV_j}\}$ from repository Inputs $ID^*_{UR_i}, PW^*_{UR_i}$, Imprint Bio^*_i ISIMC computes: $k^*_i = Rep(Bio^*_i, \tau_i)$ $(\Psi_{UR_i} \| \varphi_{UR_i})$ $= H_6(ID^*_{UR_i} \| H_0(PW^*_{UR_i} \| n_0 \| k^*_i))$ $\oplus \beta_{UR_i}$ $Auth^*_{UR_i} = H_2(\Psi_{UR_i} \| \varphi_{UR_i})$ $Auth^*_{UR_i} \stackrel{?}{=} Auth_{UR_i}$, if not equal, rejects Then, ISIMC selects $n_1, n_2 \in Z^*_q$ $K = n_1 . [Pub_{GIDSP} - H_0(ID_{SRV_j}) . Pub_{SRV_j}]$ $M_{i,j} = n_1 . Pub_{SRV_j}$ $EK_i = n_2 . P$ $\sigma_{UR_i} = \varphi_{UR_i} + n_2 \, mod \, q$ $E_{UR_i} = H_3(K) \oplus (ID_{UR_i} \| \Psi_{UR_i})$		
	$\{M_{i,j}, EK_i, E_{UR_i}, \sigma_{UR_i}\}$ $\rightarrow$	
		Computes: $K^* = \sigma_{SRV_j} . M_{i,j}$ $H_3(K^*) \oplus E_{UR_i} = (ID_{UR_i} \| \Psi_{UR_i})$ Verifies $\sigma_{UR_i} . P =$ $H_1(ID_{UR_i} \| \Psi_{UR_i}) . Pub_{GIDSP} + EK_i + \Psi_{UR_i}$, if not equal, terminates Selects $m_j \in Z^*_q$ Calculates $EK_j = m_j . P$ $EK_{i,j} = m_j . EK_j$ $SK_{i,j} = H_4(ID_{UR_i} \| ID_{SRV_j} \| EK_{i,j} \| K^*)$ $MAC = H_5(ID_{UR_i} \| EK_{i,j} \| K^* \| SK_{i,j})$
	$\{EK_j, MAC\}$ $\leftarrow$	
ISIMC calculates: $EK_{i,j} = n_2 . EK_j$ $SK_{i,j} = H_4(ID_{UR_i} \| ID_{SRV_j} \| EK_{i,j} \| K)$ $MAC^* = H_5(ID_{UR_i} \| EK_j \| K \| SK_{i,j})$ $MAC^* \stackrel{?}{=} MAC$, if not equal, rejects Otherwise: Session-key established		
	session-key $\leftrightarrow$	

The further messages exchanges between User (UR_i) and Service Organization's Server (SRV_j) will aid each side to form the transactions. For instance, UR_i requires SRV_j's information (e.g., Server's Identity) to grant the permission to it by issuing a transaction on the BC. This information is listed as SRV_j's Identity, a Request Ticket Number and a Request Ticket Timestamp (the time that the ticket number issued) which are generated by SRV_j, as well as some additional information such as the organization's name. Upon receiving the information from SRV_j, UR_i will complete the transaction formation by adding the information of the user. Afterwards, the transaction will be sent to the BC for validation and confirmation. After the transaction is sent from UR_i to the BC, the user will also send a confirmation message to the SRV_j, containing the transaction information and some additional info such as UR_i's Public Identity, an Accept Ticket Number generated by the user, the Timestamp of the acceptance message, and the additional data of UR_i.

Table 22 – Message exchanges between UR & SO

UR_i		SRV_j
		Generates RN_{SRV_j}, T_{SRV_j}
	$\{ID_{SRV_j}, RN_{SRV_j}, T_{SRV_j}, Info_{SRV_j}\}$	
	$\Leftarrow$	
Generates AN_{UR_i}, T_{UR_i} Forms the Tx and submit it to the BC		
	$\{PID_{UR_i}, AN_{UR_i}, T_{UR_i}, Info_{UR_i}\}$	
	$\Rightarrow$	
		Validates the timestamp $T_{UR_i} - T \leq \Delta T$ Queries the BC to check the AN_{UR_i}

The process continues based on the information provided by the user. SRV_j will query the BC network to let Smart Contract validate the ticket number and finally authorize the SO to provide

the service to the user. Later, GIDSP can identify UR_i as a service consumer of the designated SO by checking the ledger of the BC network. *Table 22* shows the interaction between these two parties. The communication model between User and GIDSP, where a user wants to get services from GIDSP, as well as the communication model between SO and GIDSP, where a Service Organization wants to get services from GIDSP, are identical. These communication types were described in section 3.1.3.

3.2.4. Transactions Formation

This section provides details about transactions formats and their formation based on each system phases. A Transaction (Tx) will be sent to the Blockchain (BC) network via an entity to request for applying a change to the state database [17]. Therefore, to provide a vision of how transactions are structured in our system, we will describe the transactions' structures. The very first Tx in our system is relevant to the registration of a user (Section 3.2.2). When a user registers in our system, GIDSP registers the user in the BC network and obtains its blockchain identity from the BC's CA. Then, GIDSP constructs a Tx of type *User Registration Tx* and populates its values based on the users' inputs, then sends it to the BC network for verification. This Tx consists of the user's Public Identity and a set of questions/answers exchanged between user and GIDSP at the time of registration. Note that for security purposes, this set will be encrypted by the GIDSP's public key as private data readable only by the GIDSP using its secret key (more details are provided in Section 3.2.2).

Likewise, SO must be registered with the blockchain network and obtains its credentials and from the BC's CA by the GIDSP. Also, GIDSP forms a *SO Registration Tx* consisting of SO's information such as name, type of the organization, date of the registration and will be sent to the BC network.

Table 23 - Transactions formats

Tx Type	Tx Fields	Description
UR Registration Tx (Formation by GIDSP)	Name	First name/last name of the user.
	Public ID	Public Identity of the user.
	QSTs	A set of questions generated by GIDSP.
	ANSs	A set of answers to QSTs generated by UR.
	Date	The Timestamp of the Tx.
	τ	Biometric helper parameter
SO Registration Tx (Formation by GIDSP)	ID/Name	Registered SO's identity/name.
	Type	It can be a health organization, Police, firefighters, or any type of SO.
	Date	The Timestamp of the Tx.
Grant Access Tx (Formation by users)	SO ID/Name	The id/name of the service organization.
	UR PID/Name	The public id/name of the user.
	Date	The Timestamp of the Tx.
	Access Value	This will be set to "allow" in this Tx.
Revoke Access Tx (Formation by users)	SO ID/Name	The id/name of the service organization.
	UR PID/Name	The public id/name of the user.
	Date	The Timestamp of the Tx.
	Access Value (New value)	This Tx will change the state of this field to be "deny" in order to revoke the access.

Furthermore, to grant an access to the SO by a user, a transaction of type *Grant Access Tx* will be formed and sent by the user to the BC network after the successful procedure described in Section 3.2.3. This Tx adds the SO as an authorized organization to provide a service for him/her. Lastly, a user sends a *Revoke Access Tx* to the BC in order to remove a specific SO from his/her trusted organizations list. We provide a thorough list of the transactions' fields in *Table 23* and their description. The grant access and revocation transactions fields are similar but have different values for the "access value" field that distinguishes between grant and revocation types. If this field is set to "allow", the access is equal to grant, while the "deny" value represents the revocation access of the SO. The revocation value aids the Smart Contract of the BC network to deny the access of the SO to identify the user as its users, hence SO cannot provide any services to that user.

3.2.5. API Update Phase

This phase occurs when a change applied by GIDSP in the API logic hosted in the ISIMC. This update includes software improvements, adding new features, and security patches to the logic of the ISIMC. This update happens by the operator of the ISIMC.

Chapter 4 - Implementation and Evaluation of BBGAS

In the simulation of BBGAS, we faced several limitations especially in the communication between SIM Card and Smartphone's biometric sensor that provides the real-time biometric for SIM Card. Since it is infeasible to read the real-time biometric from the Smartphone's sensor due to the security measurements of the manufacturers, we simulate the Smartphone's environment in our lab by utilizing two boards. Our Smartphone simulation consists of Board A and Board B. Board A simulates the SIM Card environment and it is connected to Board B that simulates the Smartphone in our implementation. We provide more details on this in the first section of this chapter. Next, we simulate the global data storage and our Blockchain (BC) network responsible for persisting transactions between parties. The prerequisite of this simulation is provided in the next section of this chapter, as well as the details of identities registration in the network. We also provide the Smart Contract of the BC network in the same section. In the last section of this chapter, we provide details on security of the system and how we achieve the privacy of user. We utilize techniques such as authentication before each communication and data encryption at the storage level such as securing the biometric in the SIM Card and the data in the state database of the BC network.

4.1. Smartphone Simulation Environment

In this section, we provide details on the simulation of the Smartphone environment. We used Board A to simulate the SIM card, which stores the registered user's fingerprint image during the enrollment phase. Board B imitates the Smartphone's Trusted Execution Environment (TEE), where the Fingerprint Matching algorithm executes. This board is also directly connected to a

Fingerprint Sensor (FPS) that provides the real-time fingerprint image of the current user, who wants to use our system. *Figure 16* presents how components of this environment are connected.

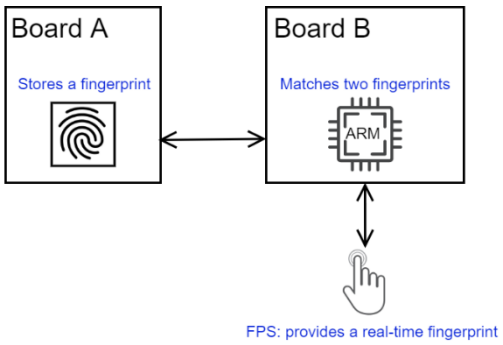


Figure 16 - Simulation Model

4.1.1. Device specifications

For Board A, we used ESP8266 NodeMCU ESP-12E [99]. This board is equipped with 64 KB of RAM, 4MB of Flash Memory, and the Microcontroller is Tensilica 32-bit. It also supports UART for accessible communication. The first reason behind our selection is that we utilize MicroPython firmware for programming this board using Python. Since ESP8266 is within the limited MicroPython supported boards [100], we have decided to utilize this board that was available in our toolbox. Other supported boards are Pyboard, STM32, Raspberry Pi RP2040, and TI CC3200 boards. The second reason for this choice is that this board provides a non-volatile memory for storing permanent data essential in our simulation and it was available in our toolbox. Board B is Raspberry Pi Model 3 B [101], equipped with a Broadcom BCM2837 processor, 1GB of RAM, and 64GB of removable flash memory. We use this board because it provides a well-documented and easy setup environment for educational purposes. Our Fingerprint Sensor (FPS) could be of any type similar to Adafruit optical sensor [102] since the Python library which we are using for fetching data from the sensor is compatible with this type. However, to connect FPS and Board B, we need a USB to UART converter. In our case, we have chosen the CP2102 6-in-1 module [103]

to implement the communication link. *Table 24* shows the connection map from FPS to CP2102. Board A and B also communicate via UART connection since both support this type.

Table 24 - USB-to-UART converter connection

FPS wire color	CP2102 pins
White	TXD
Green	RXD
Black	GND
Red	5V

Figure 17 presents the simulated working environment.

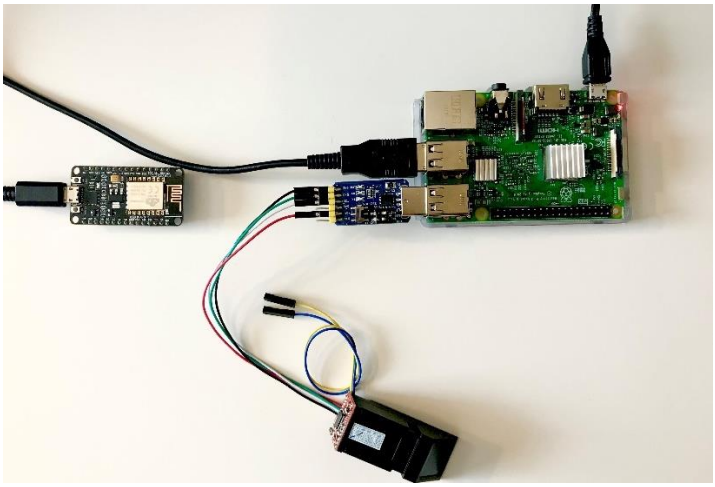


Figure 17 - Smartphone Simulation Environment

4.1.2. ESP8266 NodeMCU ESP-12E setup

This board utilizes MicroPython firmware [104]. The setup process of the firmware is followed as described in [105] and requires downloading the firmware from the MicroPython website and flash the binary file to this board via uPyCraft IDE.

The process of sending the fingerprint file to ESP8266 is done with uPyLoader accessible from [106]. On the other hand, to receive this file in Raspberry Pi, we had to port the code snippet to Raspbian. Therefore, the snippet related to downloading the file is a minimized version of this library, which is located in our repository[15]. It is worth mentioning that uPyLoader uses two internal snippets (named *__upload.py* and *__download.py*), which must be uploaded to the board before sending or downloading requests. The file structure of ESP8266 is shown in the appendix section of this manuscript.

4.1.3. Setting up the Raspberry Pi

To prepare the Raspberry Pi board for development, the Raspbian OS must be booted on Pi. First, we need to install the required software on the host machine to flash the SD card of the Pi. This software is called Raspberry Pi Imager, available on the Raspberry Pi website for download. We have done this step on the Ubuntu operating system. The steps are:

1. Install Raspberry Pi Imager (v1.5) on your host machine:

```
> sudo apt install rpi-imager
```

2. Run Raspberry Pi Imager:

```
> sudo rpi-imager
```

[15] https://github.com/nkhalili/bbgas-simulation *
* This repository is currently private.

3. Choose the Raspberry Pi OS of choice. We have chosen Raspberry Pi OS (32-bit) approx. 1 GB.
4. Insert the SD Card into the host machine and select it in Raspberry Pi Imager.
5. Click on Write, then select Yes.
6. After completion of this step, remove the SD Card and insert it into the Pi's slot. Then follow the on-screen wizard to install Raspbian in the board.

4.1.4. FPS Library

There are many libraries available for Adafruit Fingerprint sensors both in C and python languages. We have decided to use python implementation for fetching the real-time fingerprint image from the sensor because the rest of the emulator is written in python. The most common open-source library in Python is the PyFingerprint, available on this GitHub repository [107]. To install this library on Raspbian, we follow the detailed steps described in [108].

```
> sudo apt-get install python-fingerprint
```

To test the installation of this library, we navigate to the python-fingerprint directory, located in the "/usr/share/doc/". The "examples" directory contains several py files that we can run with our Python and test this library. Before running these tests, we require to connect our FPS device to the Pi. By running this command, we can get the port number to which our FPS is connected.

```
> ls /dev/ttyUSB*
```

4.1.5. Fingerprint Matching

After reading the stored fingerprint from the ESP8266 board and providing the user's real-time fingerprint from FPS, we can match these two images in our Raspberry Pi board. We do this by using the SourceAFIS implementation of the 1:1 matching algorithm provided in [109]. This

algorithm extracts features such as *bifurcation*, *minutiae*, and *ridge endings* from both given fingerprints and compares them together. The result would be a score that would be greater than or equal to a predefined threshold. If this condition is satisfied, then fingerprints are matched in terms of features. Although this matching system works very well, it is not errorless; and it is a probabilistic algorithm.

```
> boolean result = score >= 40
```

In this experiment, we conducted 30 matching processes, using the thumbs, little and index fingers of both left and right hands (six fingers). First, we registered the right hand's index finger, then we went through the matching process by choosing five other unregistered fingers. The matching process successfully detected the correct index finger as it was registered in the system. However, the matching process highly depends on the "threshold value" that was mentioned in first part of this section. Similarly, the False Accept Rate (FAR)[16] and the Flase Reject Rate (FRR)[17] percentage depends on the predefined threshold of the AFIS engine. As mentioned before, this value can be set during the matching process of the algorithm (see Section 4.1.5). By default, SourceAFIS recommends to use the value of 40, which produces FAR of 0.01% and a bit higher value for FRR but close to FAR [109]. It can be concluded that there is a trade-off between threshold, FAR, and FRR; the higher the threshold value is, the (exponentially) lower FAR and (slightly) higher FRR become. Similarly, a lower threshold results in a lower FRR.

[16] FAR is also known as False Positive Rate (FPR) and False Match Rate (FMR).
[17] FRR is also known as False Negative Rate (FNR) and False Non-Match Rate (FNMR).

4.1.6. Results

Table 25 represents the execution time of each part of this experiment. The complete source code is available on our GitHub repository. Moreover, a snapshot of the results has been provided in the appendix section.

Table 25 - Execution time of each module without encryption

Module	Execution Time	File Size
Fetch fingerprint from Sensor	7.75 sec.	74 KB
Fetch fingerprint from ESP8266	26.42 sec.	74 KB
Fingerprint Matching	9.63 sec.	N/A

As *Table 25* shows, the execution time of fetching the image from the ESP8266 is approximately 26.42 sec. We can decrease the execution time of this module by examining several approaches. The first approach compresses the file using algorithms such as gzip before uploading it to the board; this also requires the decompression process to have been completed before starting to match. Using the Python gzip library, we measured the following execution times for compressing and decompressing the fingerprint image (see *Table 26*) using python "time" library by getting the time differentiate of before and after execution of the code. Note that the file size of the image after compression is also shown. It is clear that the execution time of decompression is considerably shorter than compression. Gzip is based on LZ77 and Huffman coding [110] and the behavior of LZ77 in compression is to search for a match while in decompression it does not require such searching, instead it reads from the buffer directly [111].

Table 26 - Gzip results for compressing-decompressing fingerprint image

Operation	**Execution Time**	**Source File Size**	**Converted File Size**
Compression	0.221 sec.	74 KB	26 KB
Decompression	0.006 sec.	26 KB	74 KB

By applying this compression, the fetching process of the stored fingerprint image from the ESP8266 will decrease significantly from 26.42 sec to approximately 10.00 sec as shown in *Table 27*. The image of this experiment can be found in the Appendix section of this dissertation.

Table 27 - Execution time of biometric file before and after compression

Module	Fetch fingerprint from ESP8266
Execution Time before compression	26.42 sec.
Execution Time after applying compression	10.00 sec.

The second approach to decrease this time would be to convert the fingerprint image to a fingerprint template, which reduces the file size. However, this approach requires having access to the fingerprint sensor's template generation algorithm. We were unable to find the exact algorithm for our fingerprint sensor due to the security concerns of these sensors, which forces the manufacturer to not provide the algorithm publicly. However, both approaches would increase the matching execution procedure because they require additional processing to be executed before the matching process.

4.2. Global storage and Blockchain Network

This section will introduce our Blockchain (BC) network to store the data flows between entities. As such, we have decided to benefit from private-permissioned BCs that ensure data transparency. Our decision is based on the work reported in [24], and specifically on the use of a decision chart related to the selection between BC and traditional transaction management systems. Furthermore, we selected *Hyperledger Fabric* [17] among the various permissioned BCs, because it has better throughput and performance when the number of transactions increases compared to other BCs [44]; yet some performance issues are reported in [112]. This project is also backed by the Linux Foundation [113] and IBM has adopted this platform as its Blockchain-as-a-Service. In the upcoming subsections, we show the steps required to run the test network on the local machine. It is worth noting that our blockchain network runs on the test-network of Hyperledger Fabric using the Docker Desktop on a single PC and was used to check the logic of the Smart Contract.

4.2.1. Prerequisites

A list of required software is provided in [114]. However, we utilize WSL 2.0 on Windows 10 Home with Docker v.2.5.0.1 (stable) installed. We also require installing Node JS and npm package manager as we will develop our client applications in JavaScript. The next requirement is to install docker-compose v.1.27.4 on our development environment. Lastly, we need to install the essential build toolchains on WSL and install docker images required by fabric [115]:

```
> sudo apt install build-essential

> curl -sSL https://bit.ly/2ysbOFE | bash -s
```

4.2.2. Run Hyperledger Fabric on Local Machine

The first step to utilize Hyperledger Fabric is to bring up a test network on our local machine. All the steps are described in [116]. However, we described them here for the convenience of the reader and provided the codes in Appendix of this manuscript.

1. Clone the git repository of fabric samples to the local machine.
2. Navigate to the "test-network" directory and run the "network.sh" file. This will set up the network, which consists of Peers, Orderers, Certificate Authorities (CAs), and CouchDB instances as the world state databases per peers. It also creates a channel between organizations for future communications.
3. The next step is to deploy the Smart Contract (a.k.a. Chaincode) to the network, to be installed on each peer. Note that the *Update* process of the Chaincode in Hyperledger Fabric is identical to the installation approach. However, one additional part is to provide a version number (-ccv) and sequence number (-ccs) parameters to the end of the installation command.
4. At this point, we must have docker containers up and running for our network. To send transactions to the network, we have two options:

    4.1 Using Peer CLI: this requires additional configurations, described in [116].

    4.2 Develop a Client Application: we follow this more practical approach, especially in production environments. The client application provided in fabric samples is adequate to test the network. Therefore, we navigate to the Fabric samples folder and follow the commands provided in Step 4 of Appendix section.

The result of this experiment is depicted in *Figure 18*. As we can see, we have two organizations named *Org1* and *Org2*; each has one peer node participating in the network and maintaining their *Ledger*. The next container is the *Orderer* service responsible for managing

and queuing the transactions. There are also three CAs responsible for identities in the network. For instance, *ca_org1*, *ca_org2*, and *ca_orderer* are responsible for org1, org2, and orderer identity management, respectively. Moreover, in the steps above, we chose to use CouchDB for world state management. Therefore, we have two instances of this database; one per node (peer0.org1 and peer0.org2).

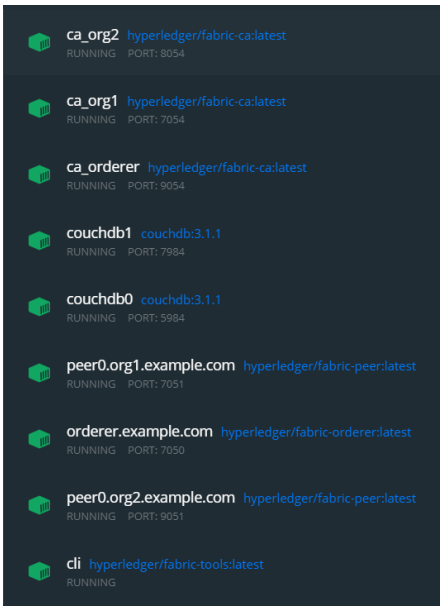


Figure 18 - Hyperledger Fabric test network containers

4.2.3. Setup Fabric Explorer

Now that we have our Blockchain network running on our pc, we wish to analyze this network. One way to inspect our network is to utilize *Blockchain Explorer* [117]. This tool provides a preliminary interface to help us obtain some data about the Blocks, Transactions, Chaincodes, and Channels. To set up this tool, we follow these steps:

1. Copy the files from this GitHub repository [117] to a new folder on our machine and include following files:

- ./connection-profile/test-network.json
- config.json
- docker-compose.yaml

2. Copy “organizations” folder from fabric samples repository to our folder
3. Open the terminal window, navigate to the root of our project, and run “docker-compose up”.

An image of this interface is attached to the Appendix section of this document.

4.2.4. Identity Registration

In Hyperledger fabric, user identities are registered by invoking functions provided by “Fabric CA Client (FCC)” that connects to the Fabric CA server of the Blockchain (BC) network [97]. Thus, to register the admin of GIDSP in the BC, the required parameters such as “enrollment Id” and “enrollment secret” (a.k.a. username and password) will be passed to the enroll function of the FCC during the *Initialization Phase*. The result of this invocation will contain the credentials of the x.509 certificate including the private key of the user that will be stored in the GIDSP’s wallet for future usages. Similarly, other parties such as User (UR) and Service Organization (SO) will be registered into the BC network by GIDSP’s admin user. Their wallets will also be initialized by this user during the *Registration Phase* of the protocol. Moreover, to improve the security of the communication channel between GIDSP and the BC network, it is crucial to enable the Transport Layer Security (TLS) on the Fabric CA server as described in [97]. The pseudocodes of the identities’ registration by utilizing “CA Client” are provided in the Appendix section [118].

4.2.5. BBGAS Chaincode

This section implements the Chaincode of our system. In Hyperledger fabric network, a Chaincode refers to the *Smart Contract* in the network responsible for transaction confirmation. Note that the

transaction's attributes are identified in section 3.2.4. Thus, in this section we provide information on the implementation of the BBGAS Chaincode, known as BbgasContract. This contract provides the following functions:

- **grantBbgas function:** this function is invoked by users during the authentication procedure of the system (see Section 3.2.3) in order to authorize an SO on the BC to provide services for them.
- **getBbgas function:** it is called by the SO who wants to validate the authenticity of an access provided by a user. This function requires the Accept Ticket Number in the previous function (grantBbgas) that is provided to the SO by the user.
- **revokeBbgas function:** in case users want to revoke the previously given access on the BC, they invoke this revocation function of the Smart Contract. It is worth noting that the history of this change is maintained on the ledger of the BC for future usage.
- **retrieveHistory function:** this provides an immutable history of the permissions provided by the user to an SO. Any given or revoked access on the BC will be returned by this function of the Smart Contract.

The JavaScript version of the "BbgasContract" is provided in the Appendix section of this manuscript.

4.2.6. Results

The evaluation results of the Blockchain network are provided in three sections: *Network statistics*, *Block details*, and *Transaction details*. The Blockchain experiment was conducted on the test network scenario of the Hyperledger fabric, described in [116]. Hence, the number of nodes has been kept at the default setting for the test network, which is one peer per organization. There is a

total number of two organizations in the default network setup of the fabric that makes the total number of nodes to be two peers.

1. **Network statistics:** these statistics are general measurements of the Blockchain network such as versions, number of working nodes, as well as number of the blocks and transactions.
    - Hyperledger Fabric version: 2.4.2
    - Number of Node(s): 2 (1 peer per each organization, namely peer0.org1, peer0.org2)
    - Number of Orderer(s): 1 named “orderer”
    - Ledger Block Height: 12
    - Number of all Transactions: 15
    - Number of Transactions in each block: Variable. The number of transactions to be collected in each block depends on the configuration of the orderer policy of the BC network. Default values of fields such as BatchTimeout and BatchSize in fabric-samples/test-network/configtx/configtx.yaml are as follows:
        - **BatchTimeout:** 2s, **BatchSize:** {MaxMessageCount: 10, AbsoluteMaxBytes: 99 MB, PreferredMaxBytes: 512 KB}).
        - Since the *BatchTimeout* is 2s, transactions will be batched into a block every 2 seconds, up to the maximum size of 99MB.
    - Number of channel(s): 1, named “myChannel”
    - World state database endpoint: http://localhost:5984/_utiles/index.html
    - Transactions’ size: Tx’s size depends on the length of the fields’ values, which in our test scenario it was approximately 94 bytes for a “grantBbgas” transaction and 54 bytes for a “denyBbgas” transaction.

2. **Block details on channel:** For the sake of simplicity, the hash values in *Table 28* are written in short forms, with four characters from the beginning and four from the end of the hashes. The complete values are provided in *Table 30*.

Table 28 – Block details of the Blockchain Network

Block number	Number of Tx	Data hash	Block hash	Previous Hash	Transactions	Block Size (KB)
11	1	30b1…9c97	d80a8…1a79	ecf3…f3b5	122f…5476	6
10	3	dcc5…c029	ecf3…f3b5	05ed…a612	486a…2ff5, 7c93…190f, c200…dcfa	15
9	2	392f…06c5	05ed…a612	3e02…1118	c901…9420, 66fd…c38b	10
8	1	356d…4418	3e02…1118	c457…a662	95d3…0ae9	6
7	1	48a9…fa99	c457…a662	936b…8c2c	0971…2e12	6
6	1	b87f…80b3	936b…8c2c	04c1…08fb	90c7…5d36	6
5	1	da1d…f3e6	04c1…08fb	e7dc…2c24	8c74…1622	7
4	1	3de2…b934	e7dc…2c24	9c0e…d9c7	91b5…aa42	5
3	1	53a7…629d	9c0e…d9c7	6dfc…3da6	2c19…d10a	5
2	1	57e1…9687	6dfc…3da6	266d…b5c4	9811…f82a	12
1	1	8cd5…22ad	266d…b5c4	2e25…c520	66cd…c702	12
0	1	bbd42…84de	2e25…c520	-	2f04…59ab	7

It is worth noting that the first six blocks (from 0 to 5) are system transactions such as lifecycles, user creations, and system configurations, which some of them are related to the network setup. Moreover, block 0 is not linked to any previous block since it is the genesis block of our network.

The number of the Block creation per hour is presented in *Figure 19*. The total number of 12 blocks has been formed in the simulation and the diagram is provided by an instance of Hyperledger Blockchain Explorer running on our machine.

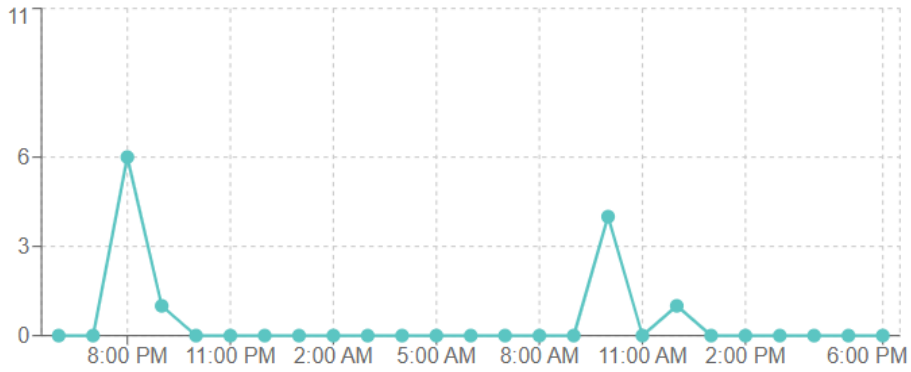


Figure 19 - Number of Blocks per hour

3. **Transaction details on channel:** The number of transactions per hour is depicted in *Figure 20* with the total number of 15 transactions. Note that the diagram below is provided by an instance of Hyperledger Blockchain Explorer on our machine.

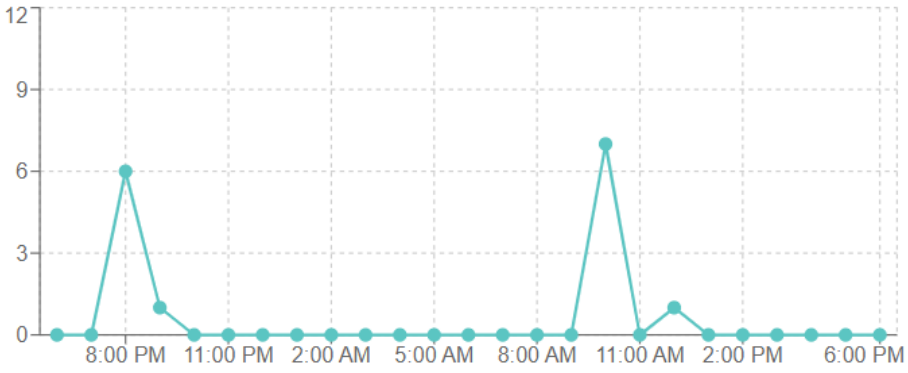


Figure 20 - Number of Transactions per hour

Table 29 represents the payloads of the transactions that were sent to the network.

Table 29 – Transactions' payloads

Tx Id	**Key**	**Payload**
122f…5476	BBGAS9	{"accessValue":"deny","date":"2022-04-25T16:12:50.401Z","orgName":"Health9","userName":"Navid"}
486a…2ff5	BBGAS7	{"userName":"Bob","orgName":"Health7","date":"2022-04-25T14:28:33.797Z","accessValue":"allow"}
7c93…190f	BBGAS9	{"userName":"Navid","orgName":"Health9","date":"2022-04-25T14:28:33.509Z","accessValue":"allow"}
c200…dcfa	BBGAS8	{"userName":"Steve","orgName":"Health8","date":"2022-04-25T14:28:32.797Z","accessValue":"allow"}
c901…9420	BBGAS5	{"userName":"John","orgName":"Health1","date":"2022-04-25T14:22:01.582Z","accessValue":"allow"}
66fd…c38b	BBGAS6	{"userName":"Bill","orgName":"Health2","date":"2022-04-25T14:22:00.946Z","accessValue":"allow"}
95d3…0ae9	BBGAS2	{"userName":"John","orgName":"Health1","date":"2022-04-25T14:20:24.912Z","accessValue":"allow"}
0971…2e12	BBGAS3	{"userName":"Bill","orgName":"Health2","date":"2022-04-25T14:19:05.825Z","accessValue":"allow"}
90c7…5d36	BBGAS1	{"userName":"test2","orgName":"health2","accessValue":"allow","date":"2022-04-21T08:10:22.522Z"}

Transactions' sources are shown in *Table 30* in order to distinguish between system transactions and business transactions that were sent by the client application.

Table 30 - Sources and complete IDs of the transactions

Tx Id	**Source**
122f7bc62a4165b202b6138f221a8f659850596c7b71695616b6c0383b3c5476	Chaincode
486a738a83916ab17a27261c5ee845b0834156a86aac9a108a32f1f2903f2ff5	Chaincode
7c933388e43b483510d7fdceb34b06f898c14584b547c6fe00e5bb98c79c190f	Chaincode
c200846064827e9d265fa312aefa45b2caca53b157e6fe6550407fbb2f13dcfa	Chaincode
c9015eeeb0ad6c6dcd65355c87db19d90f1651ec6019582c7e1df3e54ce89420	Chaincode
66fd233daa4ba8114813d893614eb5dfd5e00284402f1d7a6ba54859a7c8c38b	Chaincode
95d30f0b3ac3612bf89cf77f9d99b9f5ca11b82175f50e254dce1ca210960ae9	Chaincode
09711911c31a274617ecb2b15c755d7c93b1ca25d15276e5acfae7b6c2f52e12	Chaincode
90c78846990f08d2b57182263727fade61ced715282f8e1f1368f62c46c15d36	Chaincode
8c747e9d9efc84e03c6a0a84b091d43f382ed10ffb4849280064a90f89e91622	_lifecycle
91b5feb3e9cddd5bc50bbabae615f7166958d6d27b3941fe9d39e804069eaa42	_lifecycle
2c197076ef86f353b11b745bb017d0f2d561fd439b850c7b7cf8325db2e8d10a	_lifecycle
9811654ef6d5089bb2f6608d9679ee67511d23c9186ff55f28554daa508cf82a	CONFIG
66cd4aa68dc4beb93901724cc7cdcd3e3d0352fed889f4d2ba0f6b619442c702	CONFIG
2f0456ca4e2390c5353e0a5d7ffd5fb7c267468550f35382e8c7a656530159ab	CONFIG

The scalability of our Blockchain is similar to what Hyperledger Fabric states. According to [46], the scalability of a Blockchain depends on several factors such as number of peers, channels, transactions, and organizations. It also states that a Hyperledger fabric network can support up to

100,000 participants and the latency is fraction of a second with 200 simultaneous transactions per second[18]. In addition, the evaluation in [112] shows that Hyperledger Fabric can support up to 16 nodes with 1,000 transactions per second and the platform is not functional with more than 4 nodes when sending 10,000 transactions per second to the network.

4.3. Security and Privacy

Securing users' data has been considered seriously in our work. In this section, we provide details on the security of the users' information in our system. Although it is illusory to claim we provide a fully protected system in the cyber world, we consider preliminary steps to prevent data from being used effortlessly. As described in section 3.2.2, the user's information is stored in two locations. First location is the ISIMC that carries users' blockchain identity, biometric reproduction rate, and users' authentication identification. Second location is the Blockchain (BC) network that stores users' information during registration and authentication phases. Note that sensitive users' information on the BC are questions and answers that aid the account recovery of the user described in section 3.2.2. Since having access to these answers identifies the legitimate user, they must be confidential and available only to the original user. ISIMCs are tamper resistance computing devices that protect user's data from identity theft, yet the provided counter measurements in section 2.1.5.1 must be considered to mitigate common attacks. In addition, the registration of users is in the secured location of GIDSP, and all the communications happens via

[18] Latency: response time per transaction (in seconds), Scalability: number of serviceable participants in the network.

a secure channel. In case of a stolen ISIMC, there is no risk of exposing the user's actual biometrics data to the adversary, since we do not store any characteristics inside the ISIMC. Instead, we utilize the fuzzy extractor approach explained in section 3.2 that generates a biometric public reproduction parameter for a given biometric. Thus, the information stored on the ISIMC are safe from the adversary. Similarly, in our BC network, the registration transactions on the ledger and the registration records of the state database carry users' information that are encrypted and known only to the GIDSP at the time of user's account recovery, and thus they are secured from unauthorized access as far as the GIDSP's private key is kept hidden from the adversary.

Chapter 5 - Conclusion and Future Work

In this thesis, we designed and implemented a prototype of a global authentication system responsible for recognizing the users based on their biometrics characteristics in smart cities. It is very suitable for use in Electronic Government (E-Gov) services and smart city applications. Smartphones equipped with an Intelligent Subscriber Interface Module Card (ISIMC) that hosts the biometric file to protect the privacy of the biometric file of the user. By doing so, the file is kept separately from the rest of the system and the processes of the matching biometric files will be isolated and secured. Blockchain (BC) satisfies several attributes mentioned for E-Gov-based applications. Therefore, we introduced the design of a Blockchain-based Global Authentication System (BBGAS) that paves the way towards this goal and accomplishes the stated properties.

The BBGAS protocol consists of four system phases namely Initialization, Registration, Authentication, and API Update phases with the provided transactions formats. We found that the most significant requirement of the Authentication phase is to conduct mutual authentication. Since this process is happening inside the ISIMC, we identified a lightweight mutual authentication having the least processing loads and overheads, which can run effectively in the computation power, storage, and energy constrained devices. This process also must benefit from a lightweight cryptography approach to satisfy the security of the proposed system for transmitting data between resource-constrained devices and data stored in ISIMCs.

The next initiative of our work was to utilize a Smart Contract at the core of the system, responsible for confirming the transactions. This contract will force the regulations of the system in the authentication requests and enforce the transparency at the storage level. It is worth noting that in

this system we respect the privacy of the users by not exposing any information to other parties, thus meeting the privacy-preserving property requirement we set for the work.

In the implementation of this work, we were forced to apply some modifications to be able to simulate the environment and get the results. For instance, we utilized a board (see 4.1.1) to emulate the behavior of the ISIMC. However, in our future work we are planning to conduct the test in the more complex environments such as actual Smart Cards and Trusted Execution Environments (TEEs). A well-implemented scenario would utilize the OP-TEE [80] as part of the Global Platform Standard that relies on ARM TrustZone for virtualization and separation between secure and non-secure services. Next is the development of a Trusted Application (TA) for the Smartphone environment that isolates the biometrics verification computations of the user and the API communications for mutual authentications operational in OP-TEE. This research was completed while under the impact of the COVID-19 pandemic, which complicated our work due to laboratory closures, device accessibilities, and miscommunications. In the future we hope to expand this work to improve various aspects of this system and implement the system in more suitable environments that will provide us more accurate results. We will also publish a survey paper on lightweight authentication protocols (mentioned in Section 2.2.4) to evaluate them under same conditions and provide a more accurate comparison.

References

[1] S. Zulkarnain *et al.*, “A Review on Authentication Methods,” *Aust. J. Basic Appl. Sci.*, vol. 7, no. 5, pp. 95–107, 2013.

[2] M. Sain, O. Normurodov, C. Hong, and K. L. Hui, “A Survey on the Security in Cyber Physical System with Multi-Factor Authentication,” *Int. Conf. Adv. Commun. Technol. ICACT*, vol. 2021-Febru, pp. 1322–1329, 2021, doi: 10.23919/ICACT51234.2021.9370515.

[3] M. Raza, M. Iqbal, M. Sharif, and W. Haider, “A survey of password attacks and comparative analysis on methods for secure authentication,” *World Appl. Sci. J.*, vol. 19, no. 4, pp. 439–444, 2012, doi: 10.5829/idosi.wasj.2012.19.04.1837.

[4] and B. B. L. Bosnjak, J. Sres, “Brute-force and dictionary attack on hashed based real-world passwords,” pp. 1161–1166, 2018.

[5] A. Monzon, “Smart cities concept and challenges: Bases for the assessment of smart city projects,” *SMARTGREENS 2015 - 4th Int. Conf. Smart Cities Green ICT Syst. Proc.*, p. IS-11-IS-21, 2015.

[6] L. Yang, N. Elisa, and N. Eliot, *Privacy and security aspects of E-government in smart cities*. Elsevier Inc., 2018. ISBN:9780128150320

[7] C. Bhagavatula, B. Ur, K. Iacovino, S. M. Kywe, L. F. Cranor, and M. Savvides, “Biometric Authentication on iPhone and Android: Usability, Perceptions, and Influences on Adoption,” no. February, 2015, doi: 10.14722/usec.2015.23003.

[8] Android Open Source Project, “Android Enterprise Security White Paper,” *White Pap.*, no. September, p. 23, 2018, [Online]. Available: https://source.android.com/security/reports/Google_Android_Enterprise_Security_Whitepaper_2018.pdf

[9] Apple Inc, “Apple Platform Security,” p. 157, 2020, [Online]. Available: https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf

[10] “Mobile OS market share 2021, Statista.” https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (accessed Dec. 20, 2020).

[11] “Smartphone market share 2021, Statista.” https://www.statista.com/statistics/271496/global-market-share-held-by-smartphone-vendors-since-4th-quarter-2009/ (accessed Dec. 20, 2020).

[12] A. Atamli-Reineh, R. Borgaonkar, R. A. Balisane, G. Petracca, and A. Martin, “Analysis of trusted execution environment usage in samsung KNOX,” *SysTEX 2016 - 1st Work. Syst. Softw. Trust. Exec. Coloca. with ACM/IFIP/USENIX Middlew. 2016*, 2016, doi: 10.1145/3007788:3007795.

[13] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008, [Online]. Available:

www.bitcoin.org

[14] S. Haber and W. S. Stornetta, “How to Time-stamp a Digital Document,” pp. 437–455, 1991, [Online]. Available: https://link.springer.com/content/pdf/10.1007/3-540-38424-3_32.pdf

[15] P. N. SOFIE, “SOFIE - Secure Open Federation for Internet State of the Art Report,” no. 779984, pp. 1–67, 2018.

[16] A. Lewis, “A gentle introduction to blockchain Technology,” *Bits Blocks*, pp. 1–13, 2015, [Online]. Available: https://bitsonblocks.net/2015/09/09/a-gentle-introduction-to-blockchain-technology/#incentives

[17] E. Androulaki *et al.*, “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains,” in *Proceedings of the 13th EuroSys Conference, EuroSys 2018*, 2018, vol. 2018-Janua, p. 15. doi: 10.1145/3190508.3190538.

[18] M. Hearn and R. G. Brown, “Corda: A distributed ledger,” 2019. [Online]. Available: https://www.r3.com/reports/corda-technical-whitepaper/

[19] S. De Angelis, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, “PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain,” *CEUR Workshop Proc.*, vol. 2058, pp. 1–11, 2018.

[20] J. Nambira, “The Disruptive Blockchain: Types, Platforms and Applications,” *Texila Int. J. Acad. Res.*, no. February, pp. 20–26, 2019, doi: 10.21522/tijar.2014.se.19.02.art003.

[21] G. Zyskind, O. Nathan, and A. S. Pentland, “Decentralizing privacy: Using blockchain to protect personal data,” *Proc. - 2015 IEEE Secur. Priv. Work. SPW 2015*, pp. 180–184, 2015, doi: 10.1109/SPW.2015.27.

[22] L. Marin, M. P. Pawlowski, and A. Jara, “Optimized ECC implementation for secure communication between heterogeneous IoT devices,” *Sensors (Switzerland)*, vol. 15, no. 9, pp. 21478–21499, 2015, doi: 10.3390/s150921478.

[23] A. K. Das, S. Zeadally, and D. He, “Taxonomy and analysis of security protocols for Internet of Things,” *Futur. Gener. Comput. Syst.*, vol. 89, pp. 110–125, 2018, doi: 10.1016/j.future.2018.06.027.

[24] K. Wust and A. Gervais, “Do you need a blockchain?,” *Proc. - 2018 Crypto Val. Conf. Blockchain Technol. CVCBT 2018*, no. i, pp. 45–54, 2018, doi: 10.1109/CVCBT.2018.00011.

[25] K. Mayes and K. Markantonakis, *Smart cards, tokens, security and applications: Second edition*. 2017. ISBN:9783319505008

[26] “NXP Semiconductors History — Silicon Valley Historical Association.” https://www.siliconvalleyhistorical.org/nxp-semiconductors-history (accessed Oct. 17, 2020).

[27] Oracle, “Java Card 3 Platform Development Kit User Guide, Classic Edition,” no. April, 2018.

[28] E. S. Solutions, “Leveraging MULTOS Benefits for IoT Devices,” no. 3290642.

[29] L. Malina, P. Dzurenda, J. Hajny, and Z. Martinasek, “Assessment of Cryptography Support and Security on Programmable Smart Cards,” *2018 41st Int. Conf. Telecommun. Signal Process. TSP 2018*, pp. 269–273, 2018, doi: 10.1109/TSP.2018.8441334.

[30] C. Cachin and M. Vukolić, “Blockchain consensus protocols in the wild,” *Leibniz Int. Proc. Informatics, LIPIcs*, vol. 91, 2017, doi: 10.4230/LIPIcs.DISC.2017.1.

[31] D. Mazieres, “The Stellar Consensus Protocol - A Federated Model for internet-level consensus,” *Stellar Dev. Found.*, vol. 120, no. 42, pp. 11022–11023, 1998.

[32] “Hyperledger Sawtooth – Hyperledger Foundation.” https://www.hyperledger.org/use/sawtooth (accessed Oct. 15, 2020).

[33] A. Baliga, “Understanding Blockchain Consensus Models,” *Whitepaper*, no. April, pp. 1–14, 2017, [Online]. Available: https://www.persistent.com/wp-content/uploads/2017/04/WP-Understanding-Blockchain-Consensus-Models.pdf

[34] L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem,” *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982, doi: 10.1145/357172.357176.

[35] S. Rajsbaum, “Paxos made simple,” *ACM SIGACT News*, vol. 34, no. 4, pp. 53–56, 2003, doi: 10.1145/954092.954102.

[36] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” *Proc. 2014 USENIX Annu. Tech. Conf. USENIX ATC 2014*, pp. 305–319, 2019.

[37] M. Castro, “Practical Byzantine Fault Tolerance,” *Proc. Third Symp. Oper. Syst. Des. OSDI ’99*, no. February, pp. 1–172, 2001, [Online]. Available: http://pmg.csail.mit.edu/papers/osdi99.pdf

[38] “Ethereum Whitepaper | ethereum.org.” https://ethereum.org/en/whitepaper/ (accessed Dec. 31, 2020).

[39] “Why Cardano.” https://why.cardano.org/en/ (accessed Dec. 31, 2020).

[40] G. Greenspan, “MultiChain Private Blockchain - White Paper,” *White Pap.*, pp. 1–17, 2015, [Online]. Available: http://www.multichain.com/download/MultiChain-White-Paper.pdf

[41] H. S. Galal and A. M. Youssef, “Trustee: Full Privacy Preserving Vickrey Auction on Top of Ethereum,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11599 LNCS, pp. 190–207, 2020, doi: 10.1007/978-3-030-43725-1_14.

[42] M. J. M. Chowdhury, A. Colman, M. A. Kabir, J. Han, and P. Sarda, “Blockchain Versus Database: A Critical Analysis,” *Proc. - 17th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. 12th IEEE Int. Conf. Big Data Sci. Eng. Trust. 2018*, pp. 1348–1353, 2018, doi: 10.1109/TrustCom/BigDataSE.2018.00186.

[43] H. Saini, B. Bhushan, A. Arora, and A. Kaur, “Security vulnerabilities in Information communication technology: Blockchain to the rescue (A survey on Blockchain Technology),” *2019 2nd Int. Conf. Intell. Comput. Instrum. Control Technol. ICICICT*

2019, pp. 1680–1684, 2019, doi: 10.1109/ICICICT46008.2019.8993229.

[44] S. Pongnumkul, C. Siripanpornchana, and S. Thajchayapong, "Performance analysis of private blockchain platforms in varying workloads," *2017 26th Int. Conf. Comput. Commun. Networks, ICCCN 2017*, 2017, doi: 10.1109/ICCCN.2017.8038517.

[45] Y. Hao, Y. Li, X. Dong, L. Fang, and P. Chen, "Performance Analysis of Consensus Algorithm in Private Blockchain," *IEEE Intell. Veh. Symp. Proc.*, vol. 2018-June, no. Iv, pp. 280–285, 2018, doi: 10.1109/IVS.2018.8500557.

[46] M. Kuzlu, M. Pipattanasomporn, L. Gurses, and S. Rahman, "Performance analysis of a hyperledger fabric blockchain framework: Throughput, latency and scalability," *Proc. - 2019 2nd IEEE Int. Conf. Blockchain, Blockchain 2019*, pp. 536–540, 2019, doi: 10.1109/Blockchain.2019.00003.

[47] M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini, and Y. Khamayseh, "Comprehensive study of symmetric key and asymmetric key encryption algorithms," *Proc. 2017 Int. Conf. Eng. Technol. ICET 2017*, vol. 2018-Janua, pp. 1–7, 2018, doi: 10.1109/ICEngTechnol.2017.8308215.

[48] J. Qiu, X. Lu, and J. Lin, "Optimal selection of cryptographic algorithms in blockchain based on fuzzy analytic hierarchy process," *2019 IEEE 4th Int. Conf. Comput. Commun. Syst. ICCCS 2019*, pp. 208–212, 2019, doi: 10.1109/CCOMS.2019.8821757.

[49] M. Agrawal and P. Mishra, "A comparative survey on symmetric key encryption techniques," *Intern. J. Comput. Sci. Eng.*, vol. 4, no. 5, pp. 877–882, 2012, [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=82397469&site=ehost-live

[50] I. Publication, "Towards a Framework for a validated content management on the collaborative Web-Blogs case," *Int. J. Comput. Sci. Issues*, vol. 8, no. 3, pp. 96–104, 2011.

[51] M. Bafandehkar, S. M. Yasin, R. Mahmod, and Z. M. Hanapi, "Comparison of ECC and RSA algorithm in resource constrained devices," *2013 Int. Conf. IT Converg. Secur. ICITCS 2013*, pp. 0–2, 2013, doi: 10.1109/ICITCS.2013.6717816.

[52] M. Stokkenes, R. Ramachandra, and C. Busch, "Biometric authentication protocols on smartphones - An overview," *ACM Int. Conf. Proceeding Ser.*, vol. 20-22-July, pp. 136–140, 2016, doi: 10.1145/2947626.2951962.

[53] U. Shafique *et al.*, "Modern Authentication Techniques in Smart Phones: Security and Usability Perspective," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 1, 2017, doi: 10.14569/ijacsa.2017.080142.

[54] P. K. Singh, N. Kumar, and B. K. Gupta, "Smart Cards with Biometric Influences: An Enhanced ID Authentication," *2019 Int. Conf. Cutting-Edge Technol. Eng. ICon-CuTE 2019*, pp. 33–39, 2019, doi: 10.1109/ICon-CuTE47290.2019.8991547.

[55] Y. Li, M. Chen, and J. Wang, "Introduction to side-channel attacks and fault attacks," *2016 Asia-Pacific Int. Symp. Electromagn. Compat. APEMC 2016*, pp. 573–575, 2016, doi: 10.1109/APEMC.2016.7522801.

[56] T. Limbasiya and N. Doshi, “An analytical study of biometric based remote user authentication schemes using smart cards,” *Comput. Electr. Eng.*, vol. 59, pp. 305–321, 2017, doi: 10.1016/j.compeleceng.2017.01.026.

[57] Setiyawan, *Smart Card Security Applications Attacks and Countermeasures*, vol. 53, no. 9. 2013. ISBN:9788578110796

[58] X. Mi, Y. Zhang, and B. Wang, “A Survey of Binary Tamper-resistance Techniques for Software Protection,” vol. 50, no. Iceeecs, pp. 460–466, 2016, doi: 10.2991/iceeecs-16.2016.96.

[59] S. Ghosh, J. D. Hiser, and J. W. Davidson, “A secure and robust approach to software tamper resistance,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6387 LNCS, pp. 33–47, 2010, doi: 10.1007/978-3-642-16435-4_3.

[60] A. Alzubaidi and J. Kalita, “Authentication of smartphone users using behavioral biometrics,” *IEEE Commun. Surv. Tutorials*, vol. 18, no. 3, pp. 1998–2026, 2016, doi: 10.1109/COMST.2016.2537748.

[61] “Digital ID.” https://developer.samsung.com/digital-id/overview.html (accessed Sep. 03, 2020).

[62] M. AlRousan and B. Intrigila, “Multi-factor authentication for e-government services using a smartphone application and biometric identity verification,” *J. Comput. Sci.*, vol. 16, no. 2, pp. 217–224, 2020, doi: 10.3844/JCSSP.2020.217.224.

[63] V. Geteloma, C. K. Ayo, and R. N. Goddy-Wurlu, “A Proposed Unified Digital Id Framework for Access to Electronic Government Services,” *J. Phys. Conf. Ser.*, vol. 1378, no. 4, 2019, doi: 10.1088/1742-6596/1378/4/042039.

[64] K. Maazouz, H. Benlahmer, and N. Achtaich, “Identification System using Mobile Device Enabled NFC,” *Int. J. Comput. Appl.*, vol. 107, no. 7, pp. 37–39, 2014, doi: 10.5120/18766-0057.

[65] A. Heichlinger and P. Gallego, “A new e-ID card and online authentication in Spain,” *Identity Inf. Soc.*, vol. 3, no. 1, pp. 43–64, 2010, doi: 10.1007/s12394-010-0041-3.

[66] B. Ying and A. Nayak, “Lightweight remote user authentication protocol for multi-server 5G networks using self-certified public key cryptography,” *J. Netw. Comput. Appl.*, vol. 131, no. December 2018, pp. 66–74, 2019, doi: 10.1016/j.jnca.2019.01.017.

[67] A. Vangala, A. K. Sutrala, A. K. Das, and M. Jo, “Smart Contract-Based Blockchain-Envisioned Authentication Scheme for Smart Farming,” *IEEE Internet Things J.*, vol. 8, no. 13, pp. 10792–10806, 2021, doi: 10.1109/JIOT.2021.3050676.

[68] I. ul haq, J. Wang, and Y. Zhu, “Secure two-factor lightweight authentication protocol using self-certified public key cryptography for multi-server 5G networks,” *J. Netw. Comput. Appl.*, vol. 161, no. November 2019, pp. 1–11, 2020, doi: 10.1016/j.jnca.2020.102660.

[69] S. Dey and A. Hossain, “Session-Key Establishment and Authentication in a Smart Home Network Using Public Key Cryptography,” *IEEE Sensors Lett.*, vol. 3, no. 4, pp. 3–6, 2019,

doi: 10.1109/LSENS.2019.2905020.

[70] W. Iqbal *et al.*, “ALAM: Anonymous Lightweight Authentication Mechanism for SDN-Enabled Smart Homes,” *IEEE Internet Things J.*, vol. 8, no. 12, pp. 9622–9633, 2021, doi: 10.1109/JIOT.2020.3024058.

[71] M. Wazid, A. K. Das, R. Hussain, G. Succi, and J. J. P. C. Rodrigues, “Authentication in cloud-driven IoT-based big data environment: Survey and outlook,” *J. Syst. Archit.*, vol. 97, no. September 2018, pp. 185–196, 2019, doi: 10.1016/j.sysarc.2018.12.005.

[72] B. Vaidya, D. Makrakis, and H. Mouftah, “Two-factor mutual authentication with key agreement in wireless sensor networks,” *Secur. Commun. Networks*, vol. 9, no. 2, pp. 171–183, 2016, doi: 10.1002/sec.517.

[73] S. Yu, A. K. Das, and Y. Park, “Comments on ‘“Alam: Anonymous lightweight authentication mechanism for SDN enabled smart homes,”’” *IEEE Access*, vol. 9, pp. 49154–49159, 2021, doi: 10.1109/ACCESS.2021.3068723.

[74] V. Kumar, M. Ahmad, D. Mishra, S. Kumari, and M. K. Khan, “RSEAP: RFID based secure and efficient authentication protocol for vehicular cloud computing,” *Veh. Commun.*, vol. 22, p. 100213, 2020, doi: 10.1016/j.vehcom.2019.100213.

[75] S. Challa *et al.*, “Secure Signature-Based Authenticated Key Establishment Scheme for Future IoT Applications,” *IEEE Access*, vol. 5, pp. 3028–3043, 2017, doi: 10.1109/ACCESS.2017.2676119.

[76] B. A. Alzahrani, K. Mahmood, and S. Kumari, “Lightweight Authentication Protocol for NFC Based Anti-Counterfeiting System in IoT Infrastructure,” *IEEE Access*, vol. 8, pp. 76357–76367, 2020, doi: 10.1109/ACCESS.2020.2989305.

[77] Q. Xie, D. S. Wong, G. Wang, X. Tan, K. Chen, and L. Fang, “Provably secure dynamic ID-based anonymous two-factor authenticated key exchange protocol with extended security model,” *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 6, pp. 1382–1392, 2017, doi: 10.1109/TIFS.2017.2659640.

[78] M. Girault, “Self-certified public keys,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 547 LNCS, pp. 490–497, 1991, doi: 10.1007/3-540-46416-6_42.

[79] “Trusty TEE | Android Open Source Project.” https://source.android.com/security/trusty (accessed Jan. 09, 2021).

[80] “Getting started — OP-TEE documentation documentation.” https://optee.readthedocs.io/en/latest/general/index.html (accessed Jun. 19, 2021).

[81] S. E.-M. C. Division, “Samsung TEEgris v4.1 - GlobalPlatform TEE Security Evaluation Secretariat Certification Report GP-TEE-2020/01-CR,” 2020.

[82] GlobalPlatform, “The Trusted Execution Environment : Delivering Enhanced Security at a Lower Cost to the Mobile Market,” no. February, p. 26, 2011, [Online]. Available: http://www.globalplatform.org/documents/GlobalPlatform_TEE_White_Paper_Feb2011.pdf

[83] “Trusty and Security Services Reference — Project ACRN^{TM} 2.6 documentation.” https://projectacrn.github.io/2.6/tutorials/trustyACRN.html (accessed Mar. 21, 2021).

[84] “Introduction · littlekernel/lk Wiki.” https://github.com/littlekernel/lk/wiki/Introduction (accessed Oct. 21, 2021).

[85] “intel/ikgt-core at trusty.” https://github.com/intel/ikgt-core/tree/trusty (accessed Oct. 05, 2021).

[86] “Security-Enhanced Linux in Android | Android Open Source Project.” https://source.android.com/security/selinux (accessed Dec. 31, 2020).

[87] “Biometrics | Android Open Source Project.” https://source.android.com/security/biometric (accessed May 21, 2021).

[88] W. Li, Y. Xia, L. Lu, H. Chen, and B. Zang, “TeeV: Virtualizing trusted execution environments on mobile platforms,” *VEE 2019 - Proc. 15th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Exec. Environ.*, pp. 2–16, 2019, doi: 10.1145/3313808.3313810.

[89] “SIM / eSIM | Samsung Semiconductor Global Website.” https://www.samsung.com/semiconductor/security/sim-esim/ (accessed Jun. 17, 2020).

[90] Certicom Research, “Standards for efficient cryptography, SEC 1: Elliptic Curve Cryptography,” *Stand. Effic. Cryptogr.*, vol. 1, no. Sec 1, pp. 1–22, 2009.

[91] F. Piper and S. Murphy, *Understanding cryptography*. 2013. ISBN:9783642041006

[92] “Greatest common divisor - Wikipedia.” https://en.wikipedia.org/wiki/Greatest_common_divisor (accessed Dec. 20, 2021).

[93] S. S. Sahoo, S. Mohanty, and B. Majhi, “A secure three factor based authentication scheme for health care systems using IoT enabled devices,” *J. Ambient Intell. Humaniz. Comput.*, vol. 12, no. 1, pp. 1419–1434, 2021, doi: 10.1007/s12652-020-02213-6.

[94] N. Li, F. Guo, Y. Mu, W. Susilo, and S. Nepal, “Fuzzy Extractors for Biometric Identification,” *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 667–677, 2017, doi: 10.1109/ICDCS.2017.107.

[95] “A Python implementation of fuzzy extractor.” https://github.com/carter-yagemann/python-fuzzy-extractor (accessed Dec. 20, 2021).

[96] M. Fischlin and J. S. Coron, “Reusable Fuzzy Extractors for Low-Entropy Distributions,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9665, pp. V–VI, 2016, doi: 10.1007/978-3-662-49890-3.

[97] “Fabric CA User’s Guide — hyperledger-fabric-cadocs main documentation.” https://hyperledger-fabric-ca.readthedocs.io/en/latest/users-guide.html#fabric-ca-server (accessed Jan. 05, 2021).

[98] J. Padgette and J. Padgette, “NIST Special Publication 800-121 Revision 2. Guide to Bluetooth Security,” *NIST Spec. Publ. 800-121 Revis. 2*, pp. 1–67, 2017.

[99] “NodeMCU ESP8266 Pinout, Specifications, Features & Datasheet.”

https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet (accessed Jan. 11, 2021).

[100] “MicroPython - Python for microcontrollers.” https://micropython.org/download/#esp32 (accessed May 17, 2021).

[101] “Buy a Raspberry Pi 3 Model B – Raspberry Pi.” https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ (accessed Jan. 11, 2021).

[102] “Fingerprint sensor : ID 751 : $49.95 : Adafruit Industries, Unique & fun DIY electronics and kits.” https://www.adafruit.com/product/751 (accessed Jan. 11, 2021).

[103] E. K. Trading, “6 in 1 Multi-function Module CP2102 USB to TTL User Manual”.

[104] “MicroPython tutorial for ESP8266 — MicroPython 1.15 documentation.” https://docs.micropython.org/en/latest/esp8266/tutorial/index.html (accessed Jan. 11, 2021).

[105] “Getting Started with MicroPython on ESP32 and ESP8266 | Random Nerd Tutorials.” https://randomnerdtutorials.com/getting-started-micropython-esp32-esp8266/ (accessed Jan. 13, 2021).

[106] “BetaRavener/uPyLoader: File transfer and communication tool for MicroPython boards.” https://github.com/BetaRavener/uPyLoader (accessed Mar. 12, 2021).

[107] “bastianraschke/pyfingerprint: Python library for ZhianTec fingerprint sensors (e.g. ZFM-20, ZFM-60).” https://github.com/bastianraschke/pyfingerprint (accessed Feb. 02, 2021).

[108] “How to use a Raspberry Pi Fingerprint Sensor for Authentication.” https://tutorials-raspberrypi.com/how-to-use-raspberry-pi-fingerprint-sensor-authentication/ (accessed Feb. 02, 2021).

[109] “SourceAFIS for Java - SourceAFIS.” https://sourceafis.machinezoo.com/java (accessed Feb. 03, 2021).

[110] “gzip - Wikipedia.” https://en.wikipedia.org/wiki/Gzip (accessed Dec. 15, 2021).

[111] I. M. Pu, “Dictionary-based compression,” in *Fundamental Data Compression*, 2006, pp. 117–144. doi: 10.1016/b978-075066310-6/50010-6.

[112] A. A. Monrat, O. Schelen, and K. Andersson, “Performance Evaluation of Permissioned Blockchain Platforms,” *2020 IEEE Asia-Pacific Conf. Comput. Sci. Data Eng. CSDE 2020*, 2020, doi: 10.1109/CSDE50874.2020.9411380.

[113] Hyperledger, “An Introduction to Hyperledger [White Paper],” *Hyperledger main page*, p. 33, 2020, Accessed: Apr. 20, 2021. [Online]. Available: https://www.hyperledger.org/wp-content/uploads/2018/07/HL_Whitepaper_IntroductiontoHyperledger.pdf

[114] “Prerequisites — hyperledger-fabricdocs main documentation.” https://hyperledger-fabric.readthedocs.io/en/latest/prereqs.html (accessed May 20, 2020).

[115] “Install Fabric and Fabric Samples — hyperledger-fabricdocs main documentation.” https://hyperledger-fabric.readthedocs.io/en/latest/install.html (accessed May 16, 2020).

[116] “Using the Fabric test network — hyperledger-fabricdocs main documentation.” https://hyperledger-fabric.readthedocs.io/en/latest/test_network.html (accessed May 20, 2020).

[117] “hyperledger/blockchain-explorer.” https://github.com/hyperledger/blockchain-explorer (accessed May 15, 2020).

[118] “hyperledger/fabric-samples.” https://github.com/hyperledger/fabric-samples (accessed Jan. 03, 2021).

Appendix

1. Codes require in steps of Section 4.2.2 are provided below:

   Step 1:

   ```
   > git clone git@github.com:hyperledger/fabric-samples.git
   ```

   Step 2:

   ```
   > ./network.sh up -ca -s couchdb createChannel
   ```

   Step 3:

   Installation command:

   ```
   >./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-javascript -ccl javascript
   ```

   Update command:

   ```
   >./network.sh … -ccv 1.2 -ccs 2
   ```

   Step 4:

   ```
   > cd asset-transfer-basic\application-javascript && npm install && node app.js
   ```

2. Codes related to identities registration mentioned in Section 4.2.4 is provided below:

GIDSP's administrator registration in BC:

```
function adminRegistration(client, wallet, adminMspId) {
const registeration= client.enroll(adminId, adminPassword);
const certificate = {
      credentials: {
        certificate: registeration.certificate,
        privateKey: registeration.key.toBytes(),
```

```
      },
mspId: adminMspId,
      type: 'X.509',
    };
wallet.put(adminId, certificate);
}
```

User and SO registration in BC:

```
function userRegistration(client, wallet, userMspId, userId, organization) {
const adminId='admin';
const userRole='client';
const adminIdentity=wallet.get(adminId);
const admin=provider.getUser(adminIdentity, adminId);
const userSecret= client.register(organization, userId, userRole, admin);
const registeration= client.enroll(userId, userSecret);
const certificate = {
      credentials: {
        certificate: registeration.certificate,
        privateKey: registeration.key.toBytes(),
      },
mspId: userMspId,
      type: 'X.509',
    };
wallet.put(userId, certificate);
}
```

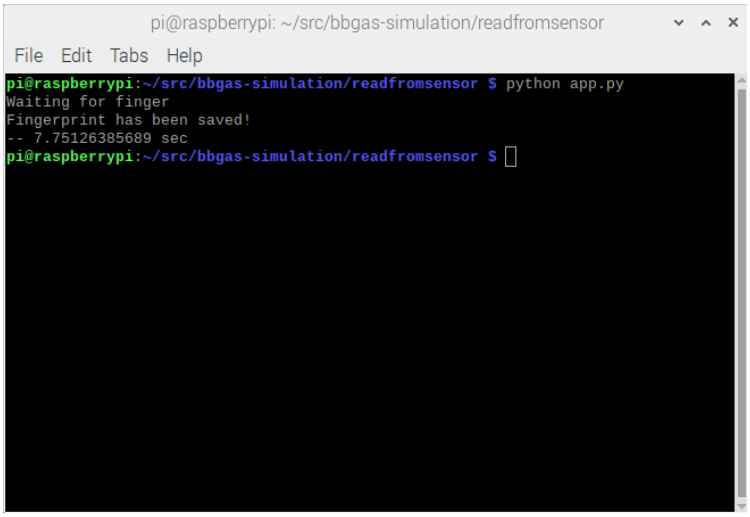
3. The code mentioned in Section 4.2.5 is provided below:

```
// Reference: https://github.com/hyperledger/fabric-samples
class BbgasContract extends Contract {

    async bbgasExists(ctx, bbgasId) {
        const buffer = await ctx.stub.getState(bbgasId);
        return !!buffer && buffer.length > 0;
    }

    // Grant access
```

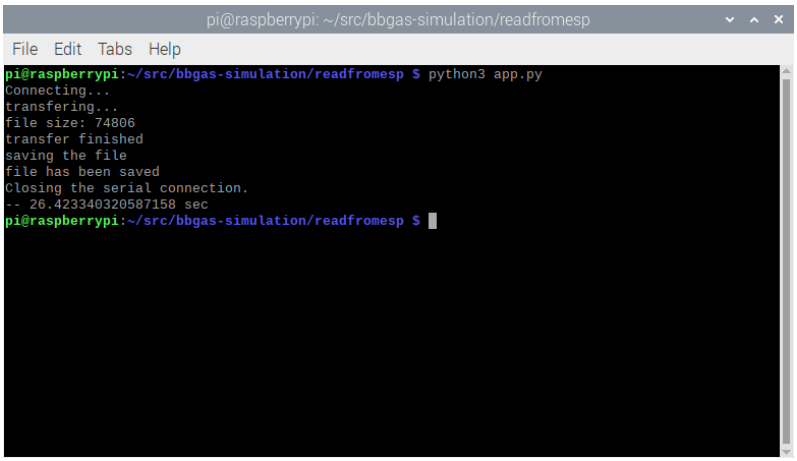
```
    async grantBbgas(ctx, bbgasId, userName, orgName) {
        const exists = await this.bbgasExists(ctx, bbgasId);
        const now = new Date().toISOString();
        let asset;
        if (exists) {
            const accessAsBytes = await ctx.stub.getState(bbgasId);
            if (!accessAsBytes || accessAsBytes.length === 0) {
                throw new Error(`The bbgas ${bbgasId} does not exist`);
            }
            asset = JSON.parse(accessAsBytes.toString());
            asset.accessValue = "allow";
            asset.date = now;
        } else {
            asset = {
                userName,
                orgName,
                date: now,
                accessValue: "allow",
            };
        }
        const buffer = Buffer.from(JSON.stringify(asset));
        await ctx.stub.putState(bbgasId, buffer);
    }

    // Read access
    async getBbgas(ctx, bbgasId) {
        const exists = await this.bbgasExists(ctx, bbgasId);
        if (!exists) {
            throw new Error(`The bbgas ${bbgasId} does not exist`);
        }
        const asset = await ctx.stub.getState(bbgasId);
        return asset.toString();
    }

    // Deny access
    async revokeBbgas(ctx, bbgasId) {
        const exists = await this.bbgasExists(ctx, bbgasId);
        if (!exists) {
            throw new Error(`The bbgas ${bbgasId} does not exist`);
        }
        const accessAsBytes = await ctx.stub.getState(bbgasId);
        if (!accessAsBytes || accessAsBytes.length === 0) {
            throw new Error(`The bbgas ${bbgasId} does not exist`);
        }
        const asset = JSON.parse(accessAsBytes.toString());
```

```
        asset.accessValue = "deny";
        asset.date = new Date().toISOString();

        const buffer = Buffer.from(JSON.stringify(asset));
        await ctx.stub.putState(bbgasId, buffer);
    }

    // Reference:
https://gist.github.com/horeaporutiu/a75704ceb85b2be032d81e1498fd0905
    async retrieveHistory(ctx, key) {
        let iterator = await ctx.stub.getHistoryForKey(key);
        let result = [];
        let res = await iterator.next();
        while (!res.done) {
            if (res.value) {
                const obj = JSON.parse(res.value.value.toString("utf8"));
                result.push(obj);
            }
            res = await iterator.next();
        }
        await iterator.close();
        return result;
    }
}
```

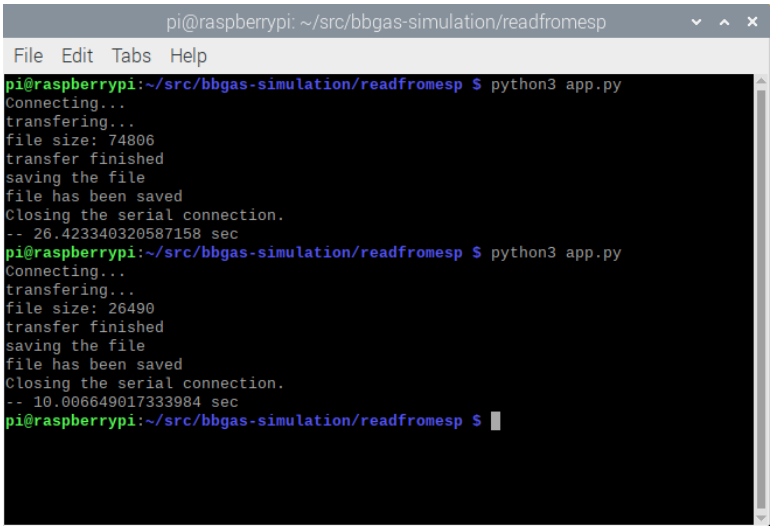
4. Images mentioned in Section 4.1.6 and 4.2.6 is provided below:



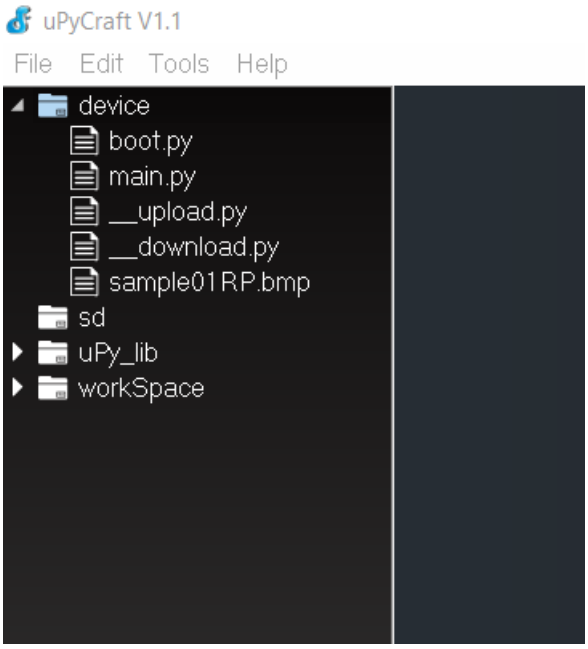
Fetch real-time fingerprint from the sensor



Read image (fingerprint.bmp) from ESP8266 Board – no compression applied

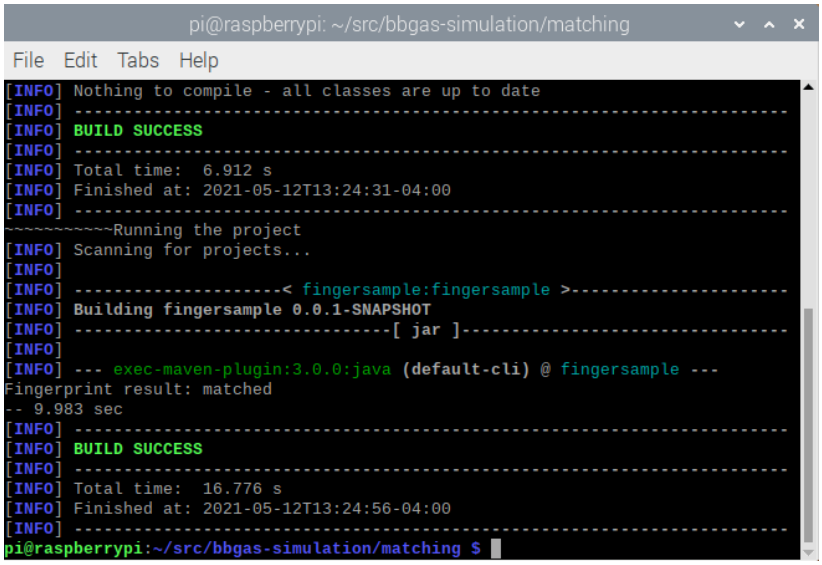


Read image (fingerprint.bmp) from ESP8266 Board – no compression vs. compression

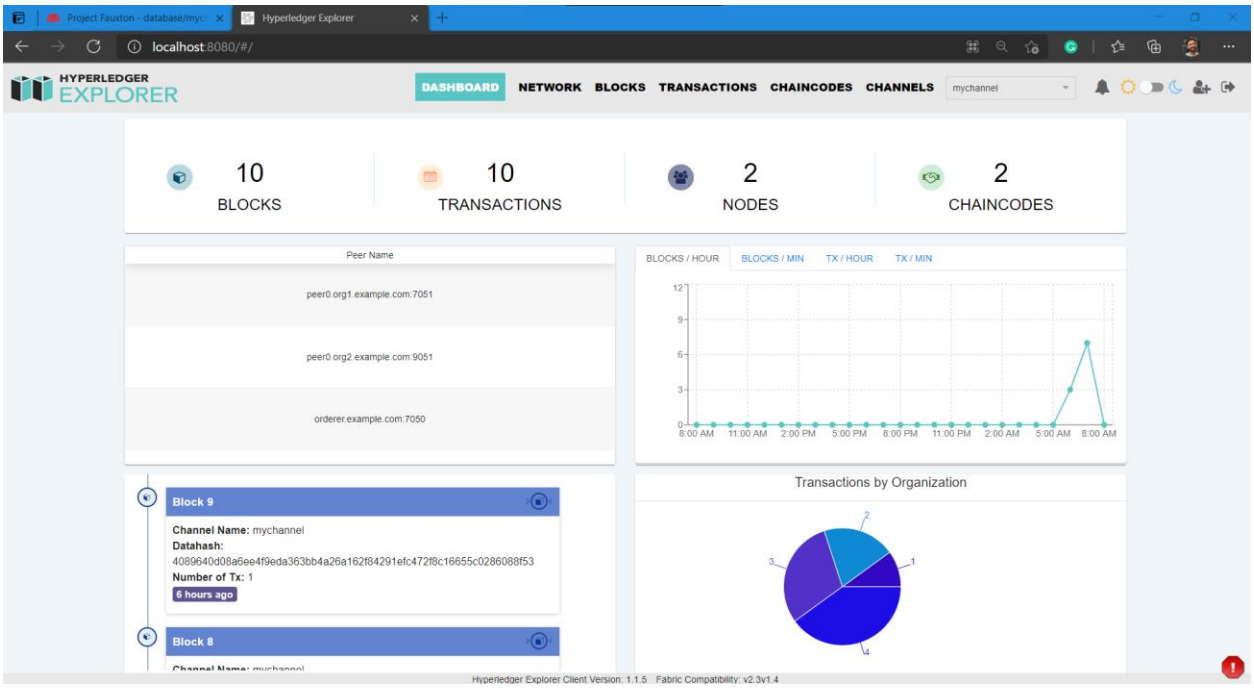


ESP8266 File Structure used uPyLoader to upload fingerprint sample to the board

(sample01RP.bmp)



Matching fingerprints on Raspberry Pi



Fabric Explorer