Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Sigmund Bernhard Berbom
Endre Medhus Mæland

Keeping connected without the Internet backbone

Master's thesis in Communication Technology
Supervisor: Yuming Jiang
Co-supervisor: Besmir Tola
June 2022

NTNU
Norwegian University of Science and Technology

Sigmund Bernhard Berbom
Endre Medhus Mæland

Keeping connected without the Internet backbone

Master's thesis in Communication Technology
Supervisor: Yuming Jiang
Co-supervisor: Besmir Tola
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Title:	Keeping connected without the Internet backbone
Students:	Endre Medhus Mæland Sigmund Bernhard Berbom

Problem description:

Most digital communication today relies on central entities to provide secure communication. In situations where the Internet is not available, due to lack of reception in remote areas, natural disasters destroying network infrastructure, or congestion due to large amounts of traffic, these central entities may not available. This causes secure communication even among users in the vicinity of each other to become a challenge.

There have been some proposed solutions to the problem, each having its drawbacks. Some require manual interaction between users to authenticate each other. Other solutions utilize mTLS for authentication but require that the users have a valid certificate signed by a central authority to freely communicate with users nearby.

In this project, we will propose a solution that utilizes the existing WiFi infrastructure to create isolated networks with users in the vicinity. We will enable users of an application to communicate securely within these networks, even when the Internet is not available. To allow secure data transport between peers in the network, the application will use mTLS. In addition, we will propose a scheme using symmetric keys to reduce the number of connections that have to be set up using mTLS.

The application will use a public key infrastructure and use certificates to ensure integrity of public keys. To allow users without a valid certificate signed by a certificate authority to communicate freely in the network we will allow use of self-signed certificates. To ensure integrity of public keys in a network allowing self-signed certificates we will design a distributed ledger that will be an agreement between the peers in the network on which public key belongs to which username.

During our master thesis we will create an instant messaging application implementing the proposed solution. We will then assess the solution by testing the application in terms of performance and security.

Date approved:	2022-02-14
Responsible professor:	Yuming Jiang, NTNU IIK
Supervisor(s):	Besmir Tola, NTNU IIK

Abstract

Most digital communication today relies on central entities to provide secure communication. However, these central entities may not be available when the Internet is unavailable due to lack of reception in remote areas, natural disasters destroying network infrastructure, or congestion due to large amounts of traffic. This causes secure communication even among users in the vicinity of each other to become a challenge. The goal of this master thesis is to design and implement a solution that enables peers within the vicinity of each other to communicate securely without a connection to the Internet backbone.

The proposed solution is threefold. First, the solution uses access points running Wi-Fi in infrastructure mode to enable users to communicate. To manage users in an offline environment, the system provides a solution based on public key infrastructure that lets all the users reach an agreement on which username belongs to which public key. This enables authentication without a third party. The system for authentication is the main contribution of the thesis. Finally, the solution lets users set up secure communication channels using mutual authentication to exchange data securely.

The proof of concept application makes it possible to measure the performance and security of the application. The results from these measurements are then discussed to evaluate the proposed solution.

Sammendrag

Store deler av dagens digitale kommunikasjon er avhengig av en eller flere sentrale enheter som kan sørge for sikker kommunikasjon. Slike enheter er ofte utilgjengelige når internett er utilgjengelig. Utilgjengeligheten kan skyldes mangel på dekning, naturkatastrofer som ødelegger infrastruktur i nettverket, eller overbelastning i nettverket grunnet mye trafikk. Dette gjør det vanskelig å sette opp sikker digital kommunikasjon selv mellom enheter som befinner seg i nærheten av hverandre. Målet med denne masteroppgaven er å designe og implementere et system som lar enheter i nærheten av hverandre kommunisere sikkert selv når internett er utilgjengelig.

Det foreslåtte systemet består av tre deler. For å opprette forbindelse bruker løsningen Wi-Fi i infrastruktur modus. For å holde oversikt over brukerne i nettverket har systemet en trygg løsning for å lagre offentlige nøkler sammen med brukernavn. Dette gjør at brukere i løsningen kan autentisere hverandre uten en tredjepart. Denne autentiseringsløsningen er hovedbidraget i masteroppgaven. Systemet har også metoder for å sette opp sikker kommunikasjon med gjensidig autentisering.

Systemet er implementert i en applikasjon som gjør det mulig å måle ytelsen og sikkerheten til systemet. Resultatene fra disse målingene brukes til å diskutere den foreslåtte løsningen.

Preface

This thesis was submitted to finalize our master's degree in Communication Technology at the Norwegian University of Science and Technology. The thesis was completed in the spring of 2022 and was based on the pre-project conducted in the fall of 2021.

We want to express our gratitude to our supervisors, Yuming Jiang and Besmir Tola, for their guidance during this master project.

Endre Medhus Mæland and Sigmund Bernhard Berbom

Contents

List of Figures

List of Tables

List of Acronyms

AES Advanced Encryption Standard.

AP Access Point.

BSS Basic Service Set.

BSSID Basic Service Set Identifier.

BTC Bitcoin.

CA Certificate Authority.

CBC Cipher Block Chaining.

CCM Counter with Cipher Block Chaining Message Authentication Code.

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance.

D2D Device-to-Device.

DAG Directed Acyclic Graph.

DH Diffie-Hellman.

DLT Distributed Ledger Technology.

DoS Denial-of-Service.

DW Discovery Window.

E2E End-to-End.

ECC Elliptic-curve cryptography.

ECDSA Elliptic Curve Digital Signature Algorithm.

ESS Extended Service Set.

ETH Ethereum.

GCM Galois Counter Mode.

GO Group Owner.

HMAC Hash-based Message Authentication Code.

IP Internet Protocol.

ITU International Telecommunication Union.

KDF Key Derivation Function.

LAN Local Area Network.

LE Ledger Entry.

MITM Man-in-the-middle.

mTLS Mutual Transport Layer Security.

P2P Peer-to-Peer.

PKI Public Key Infrastructure.

PoS Proof-of-stake.

PoW Proof-of-work.

QoS Quality of Service.

RSA Rivest–Shamir–Adleman.

SHA Secure Hash Algorithm.

SSID Service Set Identifier.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

UDP User Datagram Protocol.

WLAN Wireless Local Area Network.

WPA Wi-Fi Protected Access.

WPA2 Wi-Fi Protected Access II.

X3DH Extended Triple Diffie Hellman key exchange.

Chapter 1
Introduction

When the Internet backbone is unavailable, most Internet applications do not provide their intended services. In these scenarios, users in close proximity can deliver and receive service by establishing a Peer-to-Peer (P2P) network. However, a P2P network has other characteristics than the more common client-server architecture, requiring other security mechanisms to provide secure and authenticated communication between peers. This thesis proposes a solution for secure communication without the Internet backbone using Wi-Fi technology.

The following chapter includes a further description of the problem to be solved. It also presents the methodology used to research and validate the proposed solution.

1.1 Motivation

The evolvement of the Internet and computer systems has moved a big part of human interaction from a physical to a digital medium. We are used to always having an Internet connection. Mobile devices with an Internet connection allow us to communicate with anyone from anywhere. However, without a connection to the Internet, such services cannot be utilized even when we are within the radio range of our peers. The Internet may become unavailable due to natural disasters, accidents, traffic overload, or loss of mobile coverage. In such scenarios, a P2P network can provide connectivity between peers in the vicinity.

When data are transmitted using wireless connections, the signals between the devices are exposed to everyone within the radio range of the devices. Without proper measures, the data transmitted is exposed and can easily be manipulated. To avoid this, it is important to establish a secure connection between peers.

To establish a secure connection, confidentiality, integrity, and availability must be achieved [Sta16]. Confidentiality is achieved through encryption and ensures that an attacker eavesdropping on the channel can not read any clear-text information.

Integrity can be achieved with message authentication codes which the receiver of the data can use to discover if a message has been altered. An essential part of integrity is authentication, which is proving an identity. Authentication aids in ensuring that the connection is established with the alleged user. Finally, availability ensures the system is online and working when the users want to use the system.

One architecture that aid in secure connection establishment is Public Key Infrastructure (PKI). In a PKI, anyone can create and distribute a private-public key pair. Hence, some mechanism is needed to provide integrity of the public keys and avoid, for instance, man-in-the-middle attacks. To provide key integrity, a digital certificate can be used. A digital certificate includes a public key and often a third party's signature vouching for the key. The central trusted unit signing the certificates is called the Certificate Authority (CA). If the CA is not available, new users can not get their key signed by this central trusted unit, and a CA signature can not be used to confirm the ownership of the encryption keys. That is the case for offline networks without a CA running on the Local Area Network (LAN).

Wi-Fi is a popular technology for connecting to the Internet. It provides a radio link between a user's device and an access point. The access point can further connect the user to the Internet or connect to a LAN. Wi-Fi provides confidentiality and integrity through encryption [IEE21]. It can also provide authentication through a protocol called Wi-Fi Protected Access (WPA)-Enterprise. However, WPA-Enterprise requires an authentication server for authentication. Therefore, in a scenario where the Internet is unavailable and the central unit is unavailable on the LAN, Wi-Fi can not provide authentication.

The challenge of trusting public keys without a CA-signature and the lack of authentication in Wi-Fi raises the question of how to authenticate users in an offline Wi-Fi network.

1.2 Scope

The scope of this master thesis is to design, implement and validate a system that allows users to sign up and authenticate without using a central trusted unit. The intended scenario is an office- or school building that loses Internet connection due to an accident. The building is assumed to have multiple Wi-Fi APs connected on the same LAN.

The system should share authentication credentials, such as a certificate, between users who have signed up for the service. A user should be able to sign up to the system both with and without a connection to the Internet backbone. In addition, users should be able to verify peers' authentication credentials and prove their

identities.

The system should provide secure communication. To achieve this, well-known secure protocols such as Transport Layer Security (TLS) and Diffie-Hellman (DH) are used. These protocols are considered to be secure [JKSS12]. The system will be implemented using various tools such as Android and Kotlin. The security of these tools are not considered within the scope of this thesis and are therefore assumed to be secure.

The system should provide authentication and Device-to-Device (D2D) instant messaging between devices on the same Wi-Fi Infrastructure mode network. This allows for validation of the proposed authentication solution. Providing the means to communicate with other connection technologies such as cellular networks is out of the scope of this thesis.

1.3 Challenges

Wi-Fi with Wi-Fi Protected Access II (WPA2) can provide confidentiality and integrity through encryption and data signing. However, WPA2 does not provide authentication. To provide authentication in Wi-Fi, a protocol called WPA-Enterprise can be used. However, a limitation of WPA-Enterprise is the need for an authentication server that may not be present in an offline network. This limitation means WPA-Enterprise is not suitable for use in such networks. Without additional authentication measures, Wi-Fi does not provide authentication and can not guarantee the identity of peers.

Work has previously been done on the authentication of users in offline networks. Previous solutions have proposed systems where users sign up and receive authentication material when an Internet connection is available. If the users lose the Internet connection, the authentication material can still be used. A remaining challenge is signing up and getting authentication material when the user does not have an Internet connection. Without an Internet connection, the CA is not available to sign authentication material. Hence, new mechanisms are needed to sign up and get valid authentication material in an offline network.

A distributed authentication mechanism can solve the issue with a single central trusted unit not being available on the network. A distributed authentication mechanism should allow benign nodes to agree on an immutable record of authentication material, despite the existence of malicious nodes [Sun20]. To achieve such an agreement, Distributed Ledger Technology (DLT) could be utilized. However, most DLT, for example, Bitcoin (BTC) has resource-consuming security mechanisms tailored for financial transactions. These are secured by consensus mechanisms such

as Proof-of-work (PoW) and Proof-of-stake (PoS). The former requires a large amount of resources, which is not suitable for resource-scarce mobile phones. The latter is not suitable as an authentication process does not include financial transactions and, therefore, stakes. Other DLTs use a less resource-consuming Directed Acyclic Graph (DAG), but require transactions to verify other transactions, and the security of the network is dependent on the number of transactions. Common for all of these technologies are that they require large networks to be secure; hence they are not suitable for isolated networks.

To use a distributed solution, the users have to be able to communicate efficiently with each other. Broadcast is a solution that allows distribution of information without trusting that other users forward it. By using broadcast, the solution ensures that all peers can listen to all transmitted data. The most common transport protocols today are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Compared with TCP, UDP is a more lightweight solution. UDP does not provide reliable data transfer but provide multicast [IET17]. By making every user join the same multicast group, broadcast can be achieved in UDP. TCP on the other hand, is more comprehensive and does provide reliable data transfer. However, it does not support broadcast. Hence, to achieve a one-to-many distribution, TCP would require the network to form a mesh topology, which would not scale with an increased number of users. As there is also a need for efficient resource utilization, UDP is the preferred choice when a one-to-many distribution is needed. The disadvantages with UDP have to be compensated on the application layer.

1.4 Research questions

The challenges described in the previous section give rise to the following research questions.

- How can a user be authenticated in an offline network?
- How can users reach a consensus on each other's authentication material without a central authority to allow trusted communication among peers?
- How to provide ledger distribution reliably and efficiently using UDP?

1.5 Methodology

The project has been divided into four phases: Research, Design, Implementation, and Validation. The process has been iterative, where multiple iterations over the four phases have been done.

1.5.1 Research of background and related work

The main objective in the research phase, was to understand current research on authentication in offline networks. In addition, the aim was to find unresolved issues in the field where research contributions could be made. An overview of relevant research fields and applications looking to solve similar problems was needed to achieve this. The research was mainly done on authentication solutions used in offline and online networks and connectivity technologies suitable for offline networks.

During the research phase, multiple connection technologies were researched. Wi-Fi Infrastructure mode was selected due to its high presence in society and its efficient data transfer. However, Wi-Fi Infrastructure mode does not provide authentication without additional protocols, and existing authentication protocols do not work without an authentication server. Hence, a protocol for offline authentication is needed.

The solution needs to be decentralized as an offline network can not guarantee the presence of an authentication server. Hence, distributed databases, distributed ledgers, and blockchain technologies were researched to understand how to create systems without a central governing server.

The context for the solution was selected during the research phase. The solution should provide secure and authenticated communication between devices when the backbone Internet is unavailable. For example, an imagined scenario is a school building or office building with multiple Wi-Fi APs that has lost its Internet connection due to an accident. The solution should also work in Internet-isolated areas if mobile or stationary APs are available.

1.5.2 Design

During the design phase, the architecture of the proposed solution was designed. The system consists of a distributed ledger for keeping authentication material, a way for peers to establish a secure connection, and a system for storing symmetric keys for improved connection establishment times.

The distributed ledger contains an entry for every user in the network. A Ledger Entry (LE) contains an X.509 certificate and a username. The idea is to use concepts from distributed ledger technology to ensure a one-to-one mapping between a username and a certificate. This will guarantee a persistent relationship between username and authentication material. In addition, the distributed ledger was designed with security measures to ensure correctness and synchronization between devices in the network.

To establish a connection between devices, an architecture allowing secure D2D

communication was designed. This connectivity architecture is based on existing solutions but is improved to reduce establishment times the second time two users connect. Symmetric keys and DH ratchets are essential components in the improved solution.

1.5.3 Implementation

A proof-of-concept mobile application based on the system proposed during the design phase was developed in the implementation phase. The application was developed using Android Studio version 2020.3.1 [Goo21b] and was implemented for Android 11, API level 30 [Goo21a]. The programming language used is Kotlin [Jet21]. For advanced cryptographic functions that do not have native Kotlin support, the Bouncycastle API [The21] is used. Three Samsung Galaxy S21 and three Samsung Galaxy A71 are used to run the application. All phones are running Android 11, API level 30. Two APs, Asus AC750, and Netgear Nighthawk M1 4G LTE Mobile Router are used to create a Wi-Fi network.

1.5.4 Validation

A set of tests has been conducted and network traffic has been inspected to validate the proposed solution and to ensure the application behaves and performs as expected. The tests are conducted using Android Debug Bridge, allowing for automated and efficient conduction of multiple tests. The tests include time to set up communication paths, accept a ledger, and generate a certificate.

To validate the application's security, packets have been captured and analyzed with Wireshark to ensure they are properly encrypted.

1.6 Outline

The thesis is divided into six chapters and is structured as follows:

Chapter 1: Introduction Introduces the problem to be solved and how the solution can contribute to the research field.

Chapter 2: Background and related work Introduces the technologies, protocol, and related work needed to understand the proposed solution and design choices made in the proposed solution.

Chapter 3: Proposed solution Describes how the problem is solved using WiFi Infrastructure mode, a distributed ledger, and symmetric keys.

Chapter 4: Proof of concept application Describes how the solution is implemented in the application.

Chapter 5: Results Presents the results from the tests that have been conducted.

Chapter 6: Conclusion Concludes the thesis with final remarks.

Chapter 2
Background and related work

This chapter introduces technologies used in the proposed solution and technologies needed to understand the design choices made in the proposed solution. Relevant related and previous work on the topic is also presented.

2.1 Background

This section presents relevant technologies and concepts that have been considered or used in the proposed solution.

2.1.1 Connection technology

To establish a connection between devices, a set of protocols and technologies are needed. One popular connection technology is Wi-Fi IEEE 802.11, which specifies a set of media access control and physical layer protocols for establishing a Wireless Local Area Network (WLAN). In addition, there are alternatives to Wi-Fi for establishing a WLAN, such as Bluetooth, Zigbee, and Z-Wave. This section will present different versions of Wi-Fi and some alternatives to Wi-Fi.

Research on connection technology was conducted during the pre-project preceding this thesis [BMTJ21]. Hence, parts of the presentation from the project report are included below.

Wi-Fi infrastructure mode

Infrastructure mode is the most common mode of operation for Wi-Fi IEEE 802.11. In infrastructure mode, all communication goes through an AP. The AP is responsible for coordinating the communication on the wireless network and can work as a gateway providing a connection to the Internet. One advantage of using infrastructure mode is the AP's specialized hardware. For example, higher-power wireless radios and

antennas allow the network to cover a wider area, and specialized instruction sets allow for fast routing and forwarding.

In IEEE 802.11, a service set is a group of devices that shares the same Service Set Identifier (SSID). There are two types of services sets: Basic Service Set (BSS) and Extended Service Set (ESS). A BSS is a subgroup of the service set where the devices share the same physical-layer medium access. All devices in a BSS share the same Basic Service Set Identifier (BSSID). An ESS is a wireless network consisting of multiple APs which appear to users as a single network. This is achieved by having multiple BSS with the same logical network segment like a shared IP subnet [Gar07]. By utilizing the advantages of ESS, IEEE 802.11 can cover larger areas than the radio range of a single AP.

Wi-Fi ad-hoc

With Wi-Fi IEEE 802.11 in ad-hoc mode, it is possible to set up D2D connectivity with the original IEEE 802.11 standard. This mode of operation supports both one-to-one messaging and forwarding, which allows networks of different sizes. However, IEEE 802.11 ad-hoc mode has several drawbacks, like lack of efficient power saving and limited Quality of Service (QoS) capabilities [CGS13]. This resulted in a need for new D2D technologies, like Wi-Fi Direct and Wi-Fi Aware.

Wi-Fi Direct

Wi-Fi Direct is a technology that extends the IEEE 802.11 standard. It improves the D2D communication of IEEE 802.11 ad-hoc by introducing software APs in devices supporting Wi-Fi Direct. Wi-Fi Direct forms groups of devices, choosing one device as Group Owner (GO), to assume the responsibilities usually held by the AP [All16]. Using Wi-Fi Direct requires significant time to set up a connection [STJ19]. The GO also has a significantly higher resource consumption than the other nodes due to its special responsibility [STJ19]. In addition to putting a high load on the GO, this may lead to the GO being a bottleneck limiting the scalability of the network.

Wi-Fi Aware

Like Wi-Fi Direct, Wi-Fi Aware is an extension of the IEEE 802.11 standard [All21]. In Wi-Fi Aware, devices forms clusters where devices can advertise services supported. Two devices that want to connect can set up a direct communication path. The time it takes to set up a connection varies a lot and may be affected by a mismatch between the devices' Discovery Window (DW) [SSTJ20]. Additional security vulnerabilities may arise due to the standard operation of WiFi-Aware, making it less suitable for secure offline communication. For example, if a malicious node sends out false

synchronization beacons, a new user may not be able to access services resulting in a Denial-of-Service (DoS) attack on the network [SSTJ20].

Bluetooth mesh

Bluetooth Mesh is a communication technology that allows for many-to-many communication over Bluetooth radio. With segmentation and reassembly, Bluetooth Mesh messages can be up to 384 bytes long. The architecture is decentralized and supports multicasting by using publish/subscribe addresses. This results in greater scalability and performance compared to native Bluetooth technology. Bluetooth Mesh has some built-in security measures where the initial authentication requires a user to do some out-of-bound communication between devices [Woo20]. The technology allows for message forwarding, meaning the mesh can cover large areas, but the intermediate device distance can not exceed the Bluetooth radio range of about 10m [RY06]. The limited range makes it less suitable when a large area needs to be covered. In addition, Bluetooth has a lower data transfer rate than Wi-Fi, making it less suitable for large data transfers.

2.1.2 Carrier-sense multiple access with collision avoidance

When sending data over a wireless channel, a protocol for avoiding collisions is needed. In Wi-Fi this protocol is Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) [KR13]. In CSMA/CA devices take turns sending data one at a time. While one device is transmitting, the other devices listen to the channel. When a device has a frame ready for transmission, it listens to the channel to determine if it is idle. If the channel is idle, the device starts transferring its frame. If the channel is not idle, the device selects a random back-off time, waits, and checks again if the channel is idle.

Due to the hidden terminal problem [KR13], CSMA/CA does not always avoid collisions. If device D1 transmits data outside of the radio range of device D2, the device D2 believes the channel is idle and may start transmitting data. In this case, the channel will be wasted in the entire period of the transmission.

2.1.3 Network security

Confidentiality, integrity, and availability are essential for achieving security in computer networks. These principles are known as the CIA triad and are viewed as the fundamental objectives of information security [Sta16]. These principles are all important, but different systems will value these principles differently.

Confidentiality is to prevent unauthorized users from viewing or accessing data or services. Encryption is often used to achieve this. Encryption transforms plaintext to

ciphertext making it difficult for an attacker to read the data. Good confidentiality increases privacy as it hides sensitive information from unauthorized users.

Integrity is to prevent an unauthorized change of data. It should allow detection of malicious altering of data and system errors altering data. Integrity should also include authentication to ensure that the alleged user sent the data. Checks like hashes and signatures can be appended to the data to detect modification, duplication, deletion, or replays.

Availability ensures that a system is working correctly and provides a reliable service. If a system is not available, it cannot provide its intended services. For example, natural disasters, power outages, or DoS attacks are causes that could reduce a system's availability. In some systems, like emergency communication systems, availability is critical.

Symmetric encryption

Symmetric encryption is used to encrypt data with a single shared encryption key. All the communicating parties will share the same encryption key, which will be used for encryption and decryption. The encryption key must be shared over a secure channel. Multiple protocols exist to increase the security of the key-sharing process.

The Advanced Encryption Standard (AES) is a popular symmetric encryption algorithm. It is based on a substitution-permutation network with a fixed block size of 128 bits and a key size of 128, 192, or 256 bits. The algorithm provides security equal to its key length, meaning for a key length of 256 bits, an attack needs to perform on $2^{256} \approx 115$ quattuorvigintillion $= 115 \times 10^{75}$ combinations to break the encryption.

AES can be implemented with different modes of operations. One mode of operation is the Galois Counter Mode (GCM). GCM provides both confidentiality and integrity. It achieves integrity by computing an authentication tag that is appended as a part of the encrypted message. GCM allows for parallel processing and pipelining, making it more efficient than other modes of operations like Cipher Block Chaining (CBC), which do not support pipelining [Sta16].

Generally AES encrypts fast. Using AES hardware instructions makes it possible to achieve speeds of 1.3 cycles per byte [ADF+10]. For a computer working at 2.2GHz $= 2.2 \times 10^9$ cycles per second, AES can encrypt approximately 1.58 GB/s. The calculation is shown in Equation (2.1).

$$\frac{2.2 \times 10^9 \, cycles \, per \, second}{1.3 \times 2^{30} \, cycles \, per \, GB} \approx 1.58 \, GB/s \tag{2.1}$$

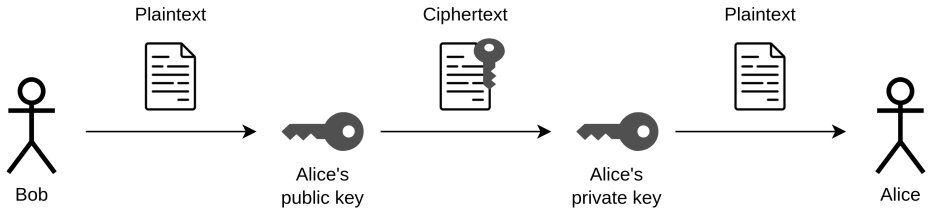


Figure 2.1: Asymmetric encryption/decryption

One of the main drawbacks of symmetric encryption is its ability to scale. Every communication pair or group need to have a unique encryption key for that pair or group. That means a total of $n(n-1)/n$ keys are required, where n is the number of pairs plus the number of groups. As the number of users in the system increases, this will cause an issue as a considerable number of keys need to be stored and handled.

Asymmetric encryption

Asymmetric encryption, also known as public-key cryptography, uses two separate keys for encryption and decryption. The key used for encryption and signature verification is called the public key, and it is exposed to the network. The key used for decryption and signature generation is called the private key, and it is kept secret [KR13].

Figure 2.1 shows a scenario where Bob wants to send a confidential message to Alice. He can use asymmetric encryption to achieve this. Bob will use Alice's public key to encrypt the message. Only Alice's private key can decrypt the message. Hence Bob knows that only Alice will be able to decrypt and read the message.

Asymmetric encryption can also provide message integrity. For example, Figure 2.2 shows a scenario where Bob wants to send a message to Alice and make sure Alice knows the message is from Bob and is unaltered. To achieve this, Bob can sign the message with his private key. When Alice receives the message from Bob, she can check the signature using Bob's public key. If the signature is correct, Alice knows Bob's private key must have generated the signature, and hence she can trust the message's integrity. Asymmetric encryption and signatures are often combined.

One popular approach to asymmetric encryption is to use Elliptic-curve cryptography (ECC). ECC is based on the elliptic curve discrete logarithm problem. With a key size of 256 bits, the scheme achieves 128 bits security, which is good compared to other asymmetric encryption schemes like Rivest–Shamir–Adleman (RSA) [BGM21].

The main drawback of asymmetric encryption is its encryption speed. For example, the authors in [LAM05] show that for a 1MB IEEE 802.11 MAC frame, AES in

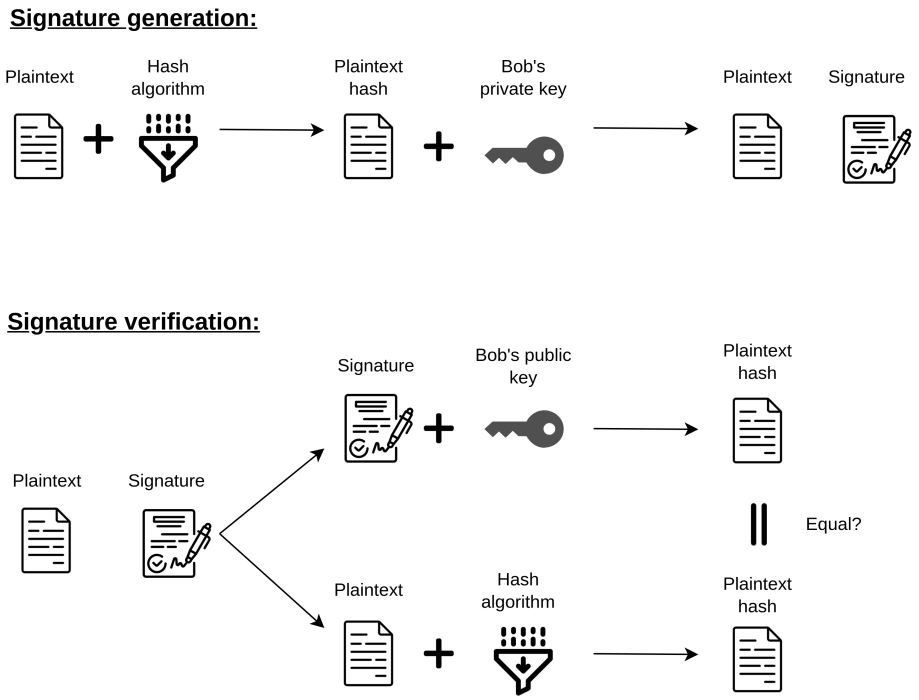


Figure 2.2: Asymmetric signature

Counter with Cipher Block Chaining Message Authentication Code (CCM) mode with 128 bits key uses only 1.404 ms to encrypt, while ECC with 283 bits key uses 35 ms [LAM05].

Public key infrastructure PKI

Anyone can generate a public/private key pair to be used in asymmetric encryption. Hence, a system to share and trust the keys is needed. A system that can achieve this is the PKI. In a PKI, a digital certificate is used as an electronic document that can prove ownership of a public key. The most common standard is X.509, which is defined by the International Telecommunication Union (ITU). An X.509 certificate includes, among other fields, a version number, a certificate serial number, validity time, unique identifier, subject's public key information, signature algorithm, and a signature [CSF+08]. A signed certificate provides a binding between the public key and a user's identity, making it possible for a peer to trust the integrity of the public key. A certificate can be self-signed or signed by a mutual trusted third party.

An authentication server can act as a trusted third party. It claims the role of a CA in a PKI, issuing and signing certificates. In a PKI, when the CA signs a

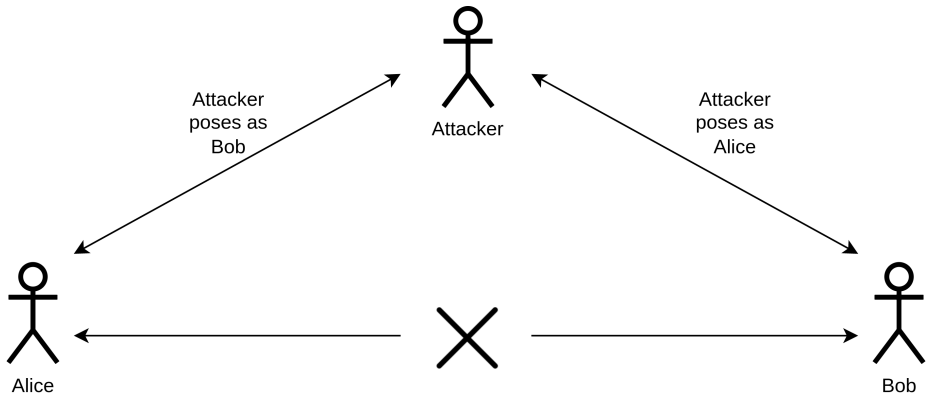


Figure 2.3: Man-in-the-middle-attack

certificate, all entities that trust the CA trust that the public key belongs to the stated user [HN17]. However, the CA may be unavailable at some times, for example, when the Internet is unavailable and the server is not on the LAN. In such scenarios, new users can not get their certificate signed by the CA, and their peers can not trust the new users.

One possible vulnerability in a PKI is a Man-in-the-middle (MITM)-attack. Figure 2.3 shows an MITM-attack. In the attack, the attacker publishes his public key to the network, claiming it belongs to another user, for example, Alice. When a third user, Bob, wants to connect with Alice, he uses what he thinks is Alice's key to establish a connection. In reality, this is the attacker's key, and the connection established is between Bob and the attacker. The attacker then establishes a connection with Alice posing as Bob. At this point, Alice and Bob believe they have established a secure communication path between each other. In reality, the attacker poses as a middleman who can read, alter and forward messages. The use of a CA-server and signed cryptographic keys mitigate this attack.

Double ratchet

Double ratchet is a concept that aims to provide key resilience, forward secrecy, and break-in recovery. To achieve this, a Key Derivation Function (KDF) is used [CY12]. A KDF is a cryptographic function that takes a secret and random KDF-key as input and returns a new key as an output. As long as the key is kept secret, the output data should be indistinguishable from random, and it should be impossible to find the input from the output. Hash-based Message Authentication Code (HMAC) [KBC97] is a hashing algorithm that meets the requirements of an KDF function [MP16a].

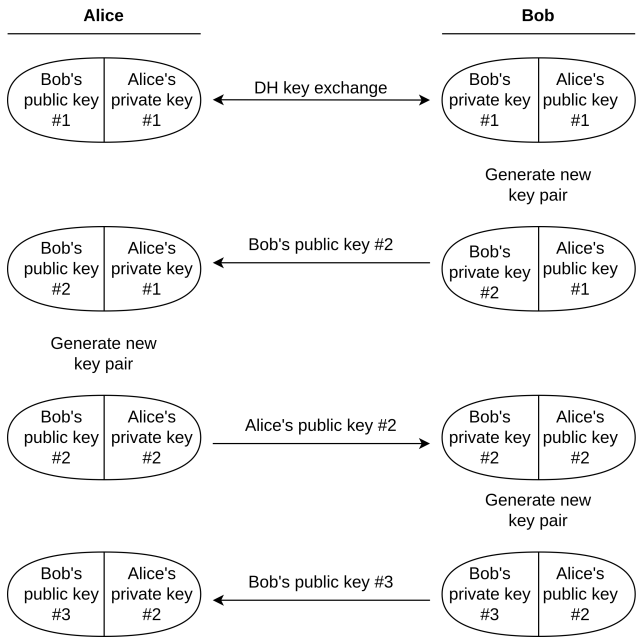


Figure 2.4: DH-ratchet

During a double ratchet session, the communicating parties keep a root chain, a sending chain, and a receiving chain. Each chain is a list of symmetric encryption keys. Every new key in the chain is the previous key hashed with a KDF [MP16a]. When sending a message, the sender should do one iteration of KDF on its sending chain, updating the encryption key. Likewise, the receiver should update its receiving chain upon receiving a message. The sending and receiving chain is reset when updating the root chain. This results in a system where every message is encrypted with a unique encryption key. This system provides forward secrecy [MP16a]. If an adversary is to capture traffic and break one encryption key, they will be able to follow the message exchange from that point, but they can not read previous messages.

DH ratchets improve the security of this system by introducing backward secrecy. On given occasions, one party can generate a new DH public key and send it to the receiver. The receiver then uses this public key to calculate a new DH secret [MP16a]. This could be viewed as half a DH key exchange. This process is shown in Figure 2.4 where parties take turns generating a new key-pair and sharing the new public key. An adversary who compromises one of the parties could learn the symmetric ratchet keys. However, after the DH ratchet round, the symmetric ratchet keys will be replaced with uncompromised ones. Consequently, the adversary is no longer able to follow the message exchange.

Mutual Transport Layer Security (mTLS)

TLS is the most important security mechanism in today's Internet [JKSS12]. TLS is a protocol that supports symmetric encryption, asymmetric encryption, and certificates to provide reliable high-end service over TCP. It is a client-server protocol where the server is responsible for providing a certificate to the client. This allows the client to verify that a mutually trusted third party has authenticated the server. As shown in Figure 2.5, the certificate and other cryptographic material are exchanged during the initial TLS handshake. The end product of the handshake is a shared encryption key to be used in symmetric encryption. For TLSv1.2 and TLSv1.3, four and two messages, respectively, are needed to establish a secure communication channel. In addition to authentication, TLS includes encryption algorithms to ensure message integrity and confidentiality [Res18]. Confidentiality is provided by agreeing on a shared secret key that will be used for one session. The messages sent in key negotiation are encrypted using the public keys. TLS provide integrity by calculating a message digest of every message sent. The level of integrity provided depends on the security of the hash function used.

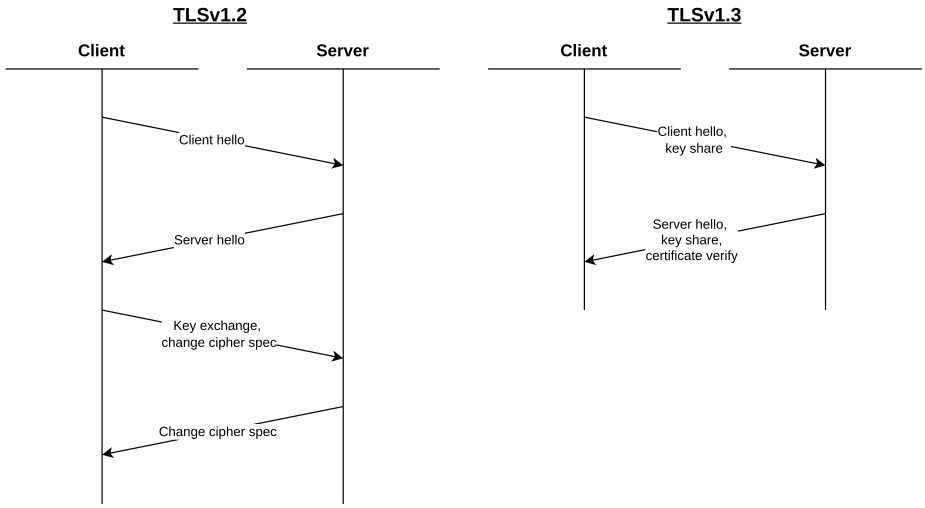


Figure 2.5: TLSv1.2 and TLSv1.3 handshake

Native TLS offers one-way authentication but can be extended to support mutual authentication with mTLS. In mutual authentication, all communicating parties must prove their identity. This is desired in verification schemes to ensure data security and verify that all parties are connected as they expect. In a P2P network, this is especially important as no dedicated servers exist, and all devices should be able to adapt to a server role and still be able to authenticate their peers. In mTLS, all parties send their certificate during the initial handshake to verify each other's identity.

2.1.4 Distributed Ledger Technology (DLT)

Many of today's web services rely on centralized data storage. Users must trust and be able to access the central entity in order to use centralized systems. In terms of availability and performance, central databases have drawbacks as well [Sun20].

A distributed ledger is an asset database that can be shared across different locations and devices. Each device holds a copy of the ledger, and changes made in one device, are reflected in the other devices. Signatures are used to authenticate changes in the ledger. Changes in the ledger can only be made according to rules agreed by the network [Adv16]. DLT assumes the existence of malicious nodes while still allowing benign nodes to agree on a shared ledger, despite the occurrence of failures [Sun20].

DLTs can be divided into three main groups: Blockchain, DAG, and Hybrid DLT. Hybrid DLT is based on both blockchains, and DAG [LFF20].

A blockchain is a chain of blocks in which each new block contains the hash of the previous block. Each block contains a collection of transactions that are confirmed when they are included in that block [Nak09]. The number of transactions that can be validated per time unit is limited because only one block can be uploaded at a time. Furthermore, because there can only be one genuine blockchain, two synchronized copies cannot be merged, even if there are no transactions that conflict. The majority of transactions in a blockchain involve a fee to encourage users to validate them. The fees of all the included transfers are awarded to the user who creates a new block.

Like a blockchain, a DAG can also store data transactions [LFF20]. While transactions in a blockchain are stored in blocks, transactions in a DAG are connected by links. Every transaction that is added to the DAG is required to verify one or more previous transactions. The transactions do thereby verify each other. An example of a DAG-based DLT is IoTA. IoTA enables secure zero-fee data and value transmission [PMC20] and has low resource utilization.

Consensus mechanism

A consensus mechanism is a fault-tolerant mechanism used to reach an agreement on a single state among nodes in a distributed network [AK21a]. These mechanisms ensure that all parties agree on the same set of valid and authenticated transactions.

PoW is a popular consensus algorithm, which is implemented by the two largest cryptocurrencies according to total market capitalization, BTC and Ethereum (ETH) [Nak09][But14]. PoW requires the nodes in the network to invest a large computational effort to create a block. In BTC a block is only valid if the hash of the header

starts with a certain number of zeros. Finding such a header requires a significant number of computations, thus limiting the rate at which new blocks can be created.

Another important consensus protocol is PoS, to which ETH will soon transition to. In PoS, the creator of the new block is chosen after a set of predefined rules, where the probability of being chosen is proportional to the number of ETH you own. The idea is that those with a large stake in the network also have the highest interest in maintaining the integrity of the blockchain.

2.2 Related work

This section presents work that has been conducted on similar topics and other solutions to the same problem. [STJ19] and [SSTJ20] where discussed in the pre-project preceding this thesis [BMTJ21]. This discussion is amended with more related work below.

2.2.1 Keeping connected when the mobile social network goes offline

In "Keeping connected when the mobile social network goes offline," the authors proposed a solution to the problem of not having an available central trusted unit [STJ19]. The solution uses Wi-Fi Direct and pre-signed certificates. The main idea was that users register for the service when the Internet is available. Then, if the Internet becomes unavailable, pre-registered users could connect using Wi-Fi Direct if they had a valid pre-signed certificate.

The main drawback of this solution is that users have to sign up for the application before losing an Internet connection. Moreover, a user that signs up without an Internet connection will not get a valid certificate and can not communicate with other users. Other shortcomings of this solution are the lack of forward secrecy, lack of backward secrecy, and high overhead.

2.2.2 Keeping connected in Internet-isolated locations

In "Keeping connected in Internet-isolated locations," the authors proposed an improvement to the system in [STJ19], allowing users to register after losing Internet connection [SSTJ20]. The system uses Wi-Fi Aware and pre-signed certificates. In addition, they included the option for peer signed certificates. The main idea is that users with a pre-signed certificate are trusted and can sign the certificates of new users.

To peer authenticate a user, a CA-certified user, P_A, will have to sign the unauthenticated users, P_B's certificate. When a peer authenticated user, P_B, wants

to connect to a new peer P_C, he must provide his keying material as well as the public key of the peer who signed his certificate. If the P_C trust P_A, he can also trust P_B's certificate. However, the proposed system does require at least one of the participants to be CA-authenticated when establishing a connection. This limits the possibility for two peer authenticated users to communicate with each other securely. In this system, two peer authenticated users can not communicate.

This solution partly solved the issue of not being able to join the system after losing connection to the Internet. However, the fact that two peer authenticated users can not communicate sets a limiting constraint on the communication. Also, this solution has some issues with fair resource allocation and performance, leading to a bad user experience.

2.2.3 Bubbles of Trust: A decentralized blockchain-based authentication system for IoT

In "Bubbles of Trust: A decentralized blockchain-based authentication system for IoT," the authors propose a blockchain-based solution for decentralized authentication of IoT devices [HHBS18]. The proposal is based on Ethereum and creates zones of trust called Bubbels. Within the Bubble, IoT devices establish trust. However, there are no mechanisms for establishing trust between different Bubbles. Furthermore, the use of Ethereum means the solution is associated with economical cost, is unsuited for real-time applications, and requires an initialization phase with a node assuming the role of a certification authority. Hence, the solution is not suitable for use cases with high user mobility.

2.2.4 Signal

Signal is a secure communication platform that supports End-to-End (E2E) encryption and offline communication. Signal's encryption scheme is useful to reduce overhead and increase security in offline communication. Open Whisper System designs the encryption scheme. The protocol uses AES256 encryption with HMAC-SHA256 for authentication. The symmetric keys are derived from an Extended Triple Diffie Hellman key exchange (X3DH) [MP16b] and is updated by using a Double Ratchet algorithm [MP16a]. The X3DH proves to the receiver that the sender owns the public key corresponding private key. The Double Ratchet algorithm provides both forward and backward secrecy while reducing the overhead needed to calculate new symmetric keys. To trust the encryption keys used and eliminate the chance of MITM attack, Signal recommends that users compare a security number derived from the shared secret on an out-of-band communication channel.

WhatsApp is a popular communication platform [Sta22] which focuses on security, especially E2E encryption [Wha22]. WhatsApp's encryption is based on Open

Whisper Systems Signaling Protocol [Wha21] described above. Bridify is another offline messaging application based on the Signaling encryption protocol [Bri22].

2.2.5 Briar

Briar is a secure messaging application designed for safe and robust communication. Briar does not rely on a centralized server and can provide offline communication over Bluetooth and Wi-Fi. For key exchange, Briar relies on an out-of-band communication where users share a secret by, for example, scanning a QR code. A Briar user has a nickname and a corresponding public key. Multiple users can use the same nickname. Briar uses sudo-random tags appended to the transferred data to identify users and choose the correct encryption key. Both sender and receiver need the mapping between the sudo-random string and the user identity [akw22].

Chapter 3
Proposed solution

This chapter describes how the proposed solution enables authenticated and secure communication without relying on a central trusted unit. As shown in Figure 3.1, the proposed solution is divided into three layers with the following responsibilities: (1) setting up a wireless communication channel, (2) handling the identities of the users in the network, enabling authentication, and (3) setting up secure communication between devices. The chapter will first provide an overview of how the solution works before going into more detail on the different layers.

3.1 Overview of the proposed solution

This section will present how applications that use a client-server model traditionally handle authentication and the server's responsibilities in such applications. This section will further show how the proposed solution handles these tasks.

3.1.1 Authentication in client-server model

In the client-server model, the server has two primary responsibilities, authenticating the client, and providing a service, as can be seen in Figure 3.2. How the server authenticates the client is discussed below.

To authenticate a client or user means to verify its identity [Onl22]. The user's identity often refers to the user's identity within a specific service. However, the user's identity can also be connected to something outside the current service, such as an email address or legal identity document. This makes it possible to create a connection between a physical person and a user of a service. To what degree a user must be able to prove their identity should be reflected in the application's security level.

The process of authenticating a client in a client-server is illustrated in Figure 3.3. To be authenticated, the client must prove that they are indeed the person

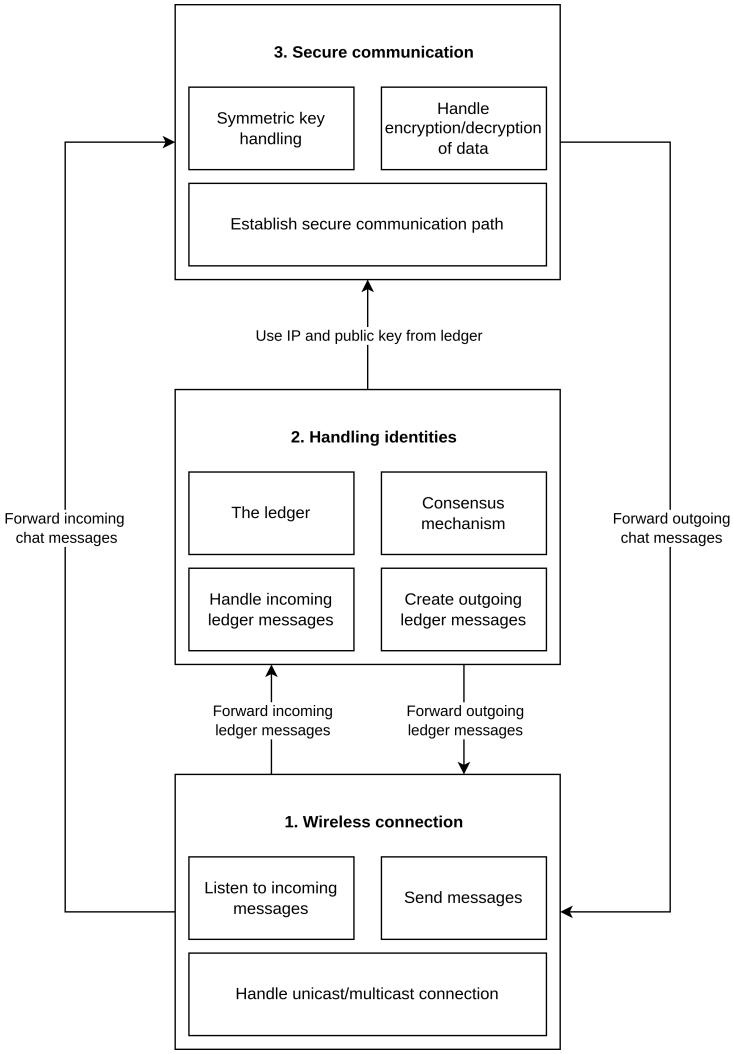


Figure 3.1: Overview of the proposed solution

who created the user in the service. The server must be able to validate this proof. Such proof can be a secret that only the client and the server know, for example, a password.

A proof to authenticate a client can also be created using public-private key pairs. The server knows the public key, and the key is associated with the user. The client can prove that he knows the private key corresponding to the public key by using cryptographic signatures. Because only the person who created the public-private key pair can generate valid signatures, a valid signature proves the client's identity.

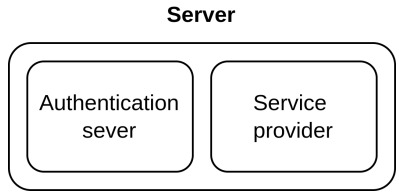


Figure 3.2: The responsibilities of the server is divided into authenticating client and poviding a service.

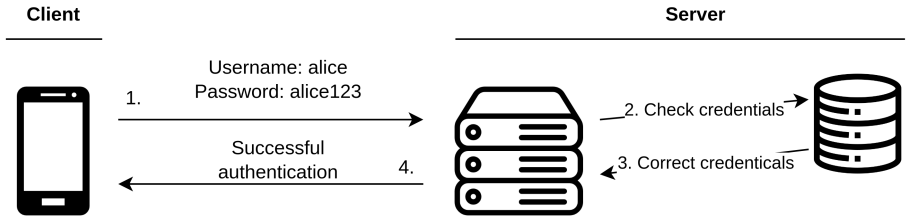


Figure 3.3: The client provides a secret known also by the server.

This method is more resource-consuming than providing a password because of the computations needed to create and validate the signatures. Because the public key is a long string of seemingly random characters, they are not easily remembered or used by humans. Hence, this authentication method is best suited when the client always uses the same device, as the private key can easily be stored on the user's device. Other authentication means exist but will not be discussed because of the lack of relevance for the proposed solution.

3.1.2 Authentication in the proposed solution

In the client-server model, the responsibility of storing authentication material, and authenticating the clients, lies with the server. The server is a trusted third party available to the clients through the Internet or on the LAN. Without access to the Internet or a trusted third party on the LAN, it is not possible to provide a central trusted unit to the users. This has several implications concerning authentication.

The responsibility of storing authentication material has to be assumed by a logical unit that all users can trust. However, the presence of a single unit trusted by all users cannot always be guaranteed in an offline network. Hence, the proposed solution does not rely on one user or server acting as the authentication server when the Internet is unavailable. Instead, the responsibility is assumed by the entire group of users as a whole. Every member of the offline network is therefore responsible for keeping a ledger. The ledger holds information on all users in the network and their authentication credentials. A user with a LE can be authenticated by their peers as

all peers hold the authentication material needed to authenticate that user. Thus, a distributed ledger with authentication material can allow peers to authenticate each other without a trusted third party.

The ledger should be synchronized between all users to achieve consensus on the users' identities and authentication materials. The system accomplishes consensus by comparing the ledgers held by all the users in the network. One consensus mechanism can be a majority decision where the ledger held by the majority of the users is accepted as the correct ledger both by new users joining the network and existing users. Chapter 3.3.5 presents the details of how this decision is made. Using the majority, the users do not have to trust any single user in the network. The trust is rather put in the network as a whole, and that the majority of users are acting benevolently.

Using shared secrets as authentication material does not work for authentication when all users use the same secret. In such a solution, the user can prove that he has access to the system, but the system can not identify different users. Therefore, the proposed solution relies on private-public keys to authenticate each other. A user can prove that they own the private key and verify that they are indeed that user to anyone in the network without exposing their secret.

A comprehensive dive into the details of how the ledger works is found in Chapter 3.3.

3.1.3 Providing the service

The second responsibility that a central trusted unit is usually trusted with is providing the actual service. For services relying on communication, such as instant messaging applications and email services, the server may act as a middleman. The server authenticates and establishes a secure connection to the users communicating and stores and forwards the data. Without a trusted third party to act as a middleman, another trusted unit has to adopt the responsibility of ensuring secure communication and providing the service.

The service delivered in the proposed solution is an instant messaging application. However, other services relying on secure communication between peers in offline networks can also apply the solution. For an instant messaging application, the service is secure direct communication between two users in the network. Direct communication does not require a middleman. Hence, no trusted third party is needed. This is possible because all the users can authenticate each other, making it possible for both parties to verify that the established communication is secure.

How the actual communication is set up is further described in Chapter 3.4.

3.2 Wireless connectivity

The first layer of the proposed solution is the wireless connectivity. The second and third layers could also work on top of a wired network, but the proposed solution focuses on a wireless connection to increase mobility and versatility. The wireless connection has to support both multicast and unicast communication. Multicast is used to ensure that every node in the network can receive messages with the intent to maintain an updated version of the ledger. Unicast is used for direct communication between peers in the network.

The proposed solution makes use of available Wi-Fi infrastructure to create LANs when the Internet is not available. Wi-Fi APs are widely available in nearly all homes and office buildings and can also be deployed easily in the case of an emergency. For users in areas without Internet availability, mobile APs can still be used to create a LAN. In addition, most mobile devices are equipped with Wi-Fi radios making Wi-Fi a technology highly available.

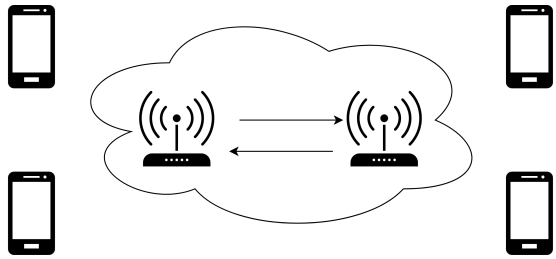


Figure 3.4: From the users perspective the network will be percevied as if there was only on AP.

In order for devices to be able to communicate, they have to be connected to the same WLAN. This WLAN may or may not have an Internet connection. To allow the solution to connect users across a large area, multiple APs can be interconnected, working together to create an extensive network. This can be done by combining them under one ESS. As can be seen in Figure 3.4, users will then perceive the network as one WLAN.

Requiring the users to be connected to the same WLAN can also serve as an extra layer of security. If the network requires the user to authenticate to get network access, the users can be viewed as more trustworthy than on an unsecured network. As only authenticated parties have access to the network, malicious users are less likely to be connected to the network. However, there is no guarantee that a network authenticated user does not have malicious intent using the proposed application. The application should also be secure on insecure networks. Hence, the ledger also provides several security measures. These will be described in the next section.

3.3 The ledger

The second layer of the proposed solution is responsible for handling identities using the ledger. As mentioned in Chapter 3.1, the ledger is responsible for enabling authentication in the network. Every user, along with their authentication material, is represented in the ledger. The ledger is an agreement among the users within the network on which users are part of the network and how to authenticate them. The following section will present the details of how the ledger works.

3.3.1 Ledger messages

To maintain a synchronized ledger among the users and distribute it to new users, the users have to be able to exchange data. When a new device joins the network, it will join and listen to a predefined multicast group. Every device using the application is a part of the same multicast group. Hence, a message sent to this group can be considered a broadcast message within the context of the solution. All messages sent concerning the ledger are sent to this group. This creates the foundation for synchronizing the ledger among all the users.

The underlying broadcast protocol is UDP. As UDP is a best-effort protocol, it can not guarantee packet delivery. To reduce the chance of lost packets, every broadcast message is sent multiple times with a given time interval between each message. The number of transmissions, as well as the time between messages, is further discussed in Chapter 5.1.

3.3.2 Ledger Entry (LE)

The ledger consists of one or more LEs. Each LE represents one user in the network and contains the data required to connect to and authenticate that user. Thus, one LE represents the identity of one user in the network. In order for a new user to join a network, they have to create a valid LE and distribute it to the other users in the network. If it is valid, the other users will add it to their ledger, making the new user a part of the network. Chapter 3.3.3 provides further details on the process of joining the network.

Each LE contains an X.509 certificate and an Internet Protocol (IP) address. The certificate contains information on both the username and public key, which are also key components of an LE. In the following section, the role of each of these pieces of information will be described.

Certificate

The most important piece of information contained in an LE is the certificate. The certificate is in the format X.509, which is a standard for public key certificates, defined by the ITU [CSF+08]. The certificate contains information about the username and public key of the user.

The certificates can be signed either by a CA or by the users themselves. The CA is a trusted third party available to the users through the Internet. To receive a certificate signed by the CA, the user has to sign up and log in to the service while the Internet is available. The user signs up using an email address as a username and a password. Before receiving a certificate from the CA, the user has to prove ownership of the email address. After signing up, the user can provide the CA with their username, password, and a public key corresponding to a private key only known to the user. If the password is correct for that username, the CA will generate, sign and send a certificate to the user based on the username and public key.

If the Internet is unavailable and a user needs to obtain a certificate, they have to generate and sign it themselves. The user will then have to generate a key pair, pick a username, generate the certificate and sign it with the private key of the generated key pair. With self-signed certificates, anyone can generate a certificate with any username. Therefore, the proposed solution implements measures ensuring that only one user can hold a specific username within a ledger.

Whether the user has a CA- or self-signed certificate is of importance when the ledger is distributed. The reason is that in order to obtain a CA-signed certificate, the user has to provide a valid email that they can prove ownership of. Hence, only one user can obtain a valid CA-signed certificate for a given email. In contrast, anyone can create a self-signed certificate with any email. Obtaining several CA-signed certificates for multiple users is time-consuming and must be done while the Internet is available. This increases the trust the system can put in a user with a CA-signed certificate.

If a user encounters one LE with a self-signed certificate and one LE with a CA-signed certificate, both with the same username, the user has to choose which one to include in their ledger. The user will then always pick the LE with a CA-signed certificate. This can happen when a new user joins the network as described in Chapter 3.3.3 or if their ledger is not up-to-date as described in Chapter 3.3.6. LEs with CA-signed certificates are always picked because they must have verified their email address. By having a CA-signed certificate, a user can always be sure that they can use it to join a ledger and not worry that their username will be taken.

Username

The username is contained in the certificate. The username is required to be an email address, and in order to obtain a certificate signed by the CA, the user has to prove ownership of the email address as well. Using an email address that the user owns is also recommended when signing up while offline because it guarantees that no one else can have a CA-signed certificate with that email address. This, in turn, guarantees that the user will not be replaced in the ledger. More details on how a user may be replaced in the ledger are described in Chapter 3.3.6.

Using an email address as a username also serves another important role. Email addresses are easy for users to read and remember. In addition, they can be easily connected to the person owning the address because they are often shared with friends, family, and colleagues. For users with a CA-signed certificate, this makes it possible not only to prove their identity within the network but also to prove that they are the person they say they are outside of the service.

Public key

The public key is, like the username, contained within the certificate, but it does also play an essential role in itself. The public key in a user's LE is what makes it possible for other users to authenticate that user. Because the corresponding private key is only known to the user who created the LE, a user only has to provide proof that they know the private key to be authenticated. This proof can only be validated by users that know the public key and is, therefore, an essential piece of information in every LE.

IP address

The IP address is also a part of every LE, but in contrast to the certificate and information contained in it, it is not used for security reasons. Instead, it is used to set up a direct communication path between peers. Therefore, to avoid having to do an IP address lookup every time two peers want to communicate, this is contained in the LE.

Because IP addresses may change, it is possible to update the IP address of an LE. This update is broadcast to all the other users, and the message is signed so that only the owner of the LE can update the IP address.

3.3.3 Joining the ledger

The ledger contains one LE for every user in the network, and every time a new user wants to join the network, a new LE has to be added. How this process works is described in the following paragraphs.

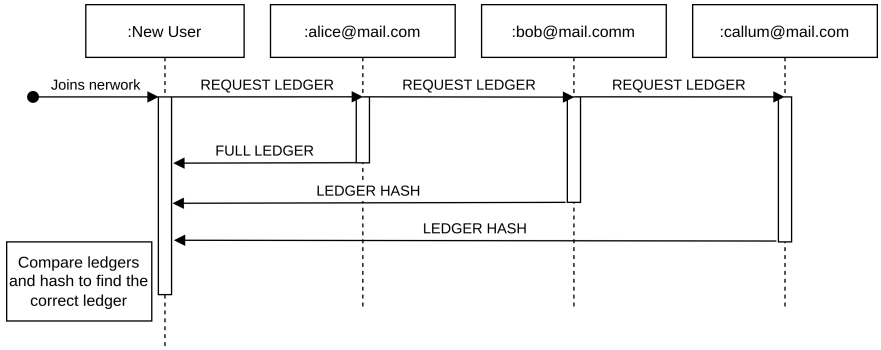


Figure 3.5: Broadcast message exchange when a new user joins the network

When a device connects to a new network, it first joins the multicast group, to which all users are listening. Secondly, it broadcasts a request for the ledger to all the users in this group. The first messages sent in Figure 3.5 shows this request. The format of both the request and it's responses are described in Chapter 4.2.1.

If the user does not receive any responses for a given amount of time, the user assumes there are no other users in the network. The amount of time the user waits is discussed in Chapter 5.1.6. The user will then proceed to create their own LE, which at this point will make up the ledger.

More often than not, the user will not be the first to join the network and will therefore receive responses from the existing users. This is shown in Figure 3.5. One of the users will respond by sending all the LEs in their ledger, while the others will send the hash of their ledger. Which of the users send the full ledger is discussed in Chapter 5.1.7. The user will accept a ledger and store it locally based on these responses. How the user decides which ledger to accept is discussed in Chapter 3.3.5.

As UDP is a best-effort protocol, packets might get lost, and all users might not receive the full ledger during a voting. A user that does not receive the full ledger will after α seconds request the ledger again. The user will pick a random user from the users that responded with the correct hash, and request the full ledger from them. If the user still does not receive the ledger within α seconds, they will request it again from another user. This is repeated until the ledger is successfully received. The value of α is discussed in Chapter 5.1.5.

After receiving the ledger, the user can create its own LE. If the user already has a stored certificate and the corresponding private key, this can be used. If not, the user will have to obtain a new certificate, as discussed in Chapter 3.3.2. Whether the user uses an existing certificate or obtains a new one, the username in the certificate cannot already exist in the ledger. The exception is if the new user has or can obtain

a CA-signed certificate for that username. If this is the case, the user will replace the conflicting LE in their ledger with their own LE. When the user then broadcasts their LE containing a CA-signed certificate, all the other users will replace the LE with that username with the LE of the new user.

When the user has successfully generated a valid LE that can be added to the ledger, the user broadcasts it to the other users. All of the users will then update their ledger accordingly.

3.3.4 Life cycle of the ledger

The first user to join a network running an application implementing the proposed solution creates a ledger. This is done simply by creating an LE. Then, for every user that joins the network, a new LE will be added, and so the ledger will continue to grow.

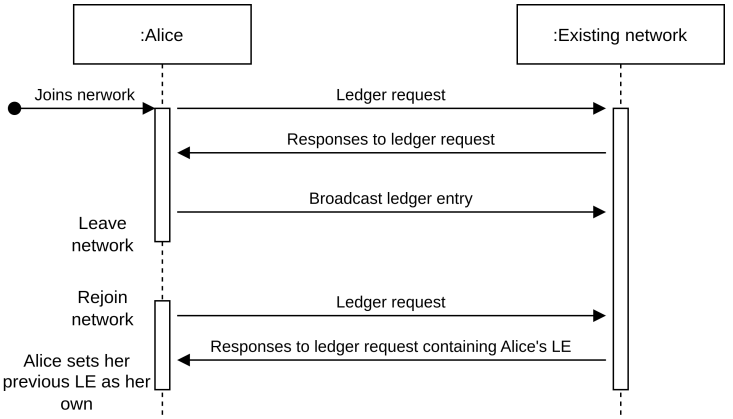


Figure 3.6: Because Alice's LE from the first time she joined the network, is still in the ledger, she does not have to create a new LE the second time she joins the same ledger.

A user who leaves the network either by quitting the application, connecting to a different network, or losing reception will delete the ledger stored locally. However, the user's certificate and keying material is stored because they may be reused in another ledger should the user join another network. The LE of the user that leaves are not removed from the ledger. Should the user rejoin the same network later in time, and the same ledger is still maintained, the user will receive a ledger containing an LE that they created. This is shown in Figure 3.6. Because the user still stores the certificate and private key, the user will discover this and will continue to use the LE already in the ledger. If the IP address of the LE does not match the user's current IP address, the new IP address will be broadcast to the network so that everyone can update the IP address of that LE.

Like a language is passed on to new generations, the ledger is passed to new users joining the network. As long as there is still a user in the network that knows the ledger, it will be passed on and continue to grow. When there are no longer any devices in the network that passes the ledger on to new users, like an ancient language, the ledger is effectively gone. The next user to join the network will not learn the previous ledger but create a new one. This ledger will continue to be passed on and grow as long as users pass it on.

3.3.5 Consensus in the ledger

Every time a new user joins the network, the user has to obtain the ledger to store it locally. Because a user does not trust any other single user, it is important that a single user can not manipulate what ledger the new user accepts. The process of learning the ledger should therefore ensure that the ledger accepted by the new user is the one held by the majority of the users.

To achieve this, all users are involved in distributing the ledger to new users. As described in Chapter 3.3.3, the process starts with the new user requesting the ledger from their peers. This request triggers a voting process. All the users in the network respond by broadcasting a response to this request. The response contains either the full ledger or a hash of the ledger. When deciding which ledger to accept, both types of responses are considered a vote. The full ledger is hashed before all hashes are compared. To ensure that each user can only vote once, the messages have to be signed, and their certificate has to be included in the message. The ledger corresponding to the most common hash will be accepted as the correct one.

When deciding what ledger is to be accepted, the one with the most votes overall is not blindly selected. The reason is that users with self-signed certificates can join the ledger and send votes. Because these certificates can easily be generated, a malicious actor could quickly generate multiple users with self-signed certificates and join a ledger. This could lead to one actor controlling the majority of users, thereby gaining the majority of the votes. Controlling the majority of users would enable the malicious actor to manipulate the ledger, which could lead to DoS attack. Gaining the majority vote in a distributed ledger, and hence being able to manipulate the ledger, is called a Sybil attack [AK21b].

To mitigate such an attack mainly votes from users with a CA-signed certificate are considered. Other votes are only considered if no votes from users with CA-signed certificates are received. That event might happen due to packet loss or when there are no users with CA-signed certificate in the network. The reason votes from users with a CA-signed certificate are valued higher is that the CA-signed certificates are harder to obtain. Firstly, they have to be obtained while the Internet is still available. Secondly, it will require the user to create an email for every certificate and validate

that they own that email address. This makes it harder for anyone to conduct a Sybil attack.

The following criteria must be met before a user accepts a ledger. The criteria are designed to provide sufficient security while still guaranteeing availability by ensuring a ledger is always accepted, regardless of the number and types of users in the network.

1. If at least two CA-certified users agree on the ledger, and they make up more than 50% of the CA-certified users in the ledger, that ledger will be accepted.
2. If α seconds have passed from the ledger was requested and at least one CA-certified user has responded with the ledger, the ledger with most votes from CA-certified users are accepted.
3. If α seconds have passed from the ledger was requested and no CA-certified users have responded with the ledger, the ledger with most votes will be accepted.

α has to be set so that users can expect to have received all the votes within that time. Further discussion on what the value should be, is found in Chapter 5.1.

The acceptance criteria are based on a majority rule, with some customization related to the user certificate type. In a majority decision, the outcome is more likely to be correct the more users participate in the decision making. More users participating makes it harder for a malicious actor or a group of malicious actors to affect decision-making. It is also less likely that a friendly user with an out-of-sync ledger will affect decision-making. Hence, the system becomes more secure as more users join the network.

3.3.6 Synchronizing the ledger

As previously mentioned, the underlying transport protocol used when sending messages concerning the ledger is best-effort. This can cause some users' ledgers to become out of sync. That can happen, for example, if messages broadcasting the LE of a new user are lost. Therefore, the proposed solution includes measures for synchronizing the ledger among users in the network.

The most important measure for keeping the ledger synchronized is that all messages concerning the ledger are broadcast. This includes requests for ledgers and all the votes sent in response to that request. Because these messages are broadcast, all the users in the network can receive and store these votes. Then, in the same way as the user joining, the other users can use these votes to see if they have an

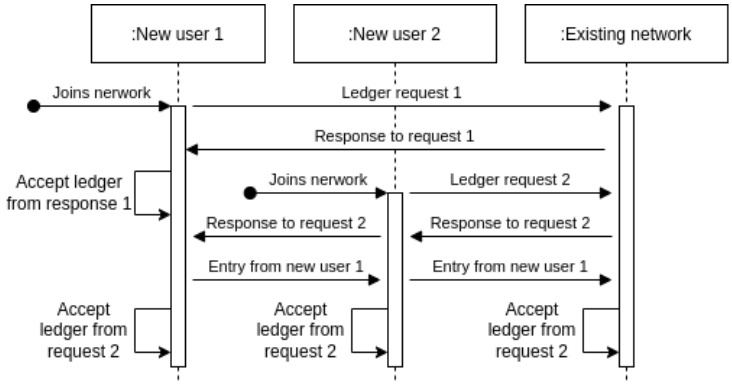


Figure 3.7: LE of new user 1 is not included in the ledger from request 2, and is therefore lost after new user 1 has registered.

updated ledger. If their ledger is not up-to-date, they can update it according to a set of rules.

If a user sees that the accepted ledger contains one or more LEs with usernames that do not exist in their ledger, they are added. If a user's ledger hold any LEs that do not exist in the accepted ledger, those LEs are not removed. There are two main reasons why LEs are never removed from a ledger. The first is that it may open the door for a ledger reset attack, which is described in detail in Chapter 3.3.7. The second reason is that deleting LEs may lead to a lost LE, which is illustrated in Figure 3.7. This can happen if two users join the network at about the same time. Because the LE of User 1 has not been broadcast yet when the responses to ledger request 2 is sent, those responses do not contain the LE of User 1. If User 1's LE is broadcast between the time when the responses to ledger request 2 are sent, and when a ledger from ledger request 2 is accepted the network will accept a ledger without User 1's LE, even though the network has already received User 1's LE. If LEs could be removed from the local ledgers if they did not exist in the accepted ledger, User 1 would be removed from the ledger and would have to sign up again. However, since they are not removed, the next time a new user requests the ledger, User 1's LE will be included in the ledger.

There could also be a conflict between some LEs in the accepted ledger and the ledger held by a user. A conflict between LEs means that two LEs with different certificates have the same username. This can happen, for instance, in the situation illustrated in Figure 3.8. What happens is that the broadcast block from User 1, message 1, is not received by all users present in the network. User 1's LE is therefore only added to User 1 and User 2's ledger. When User 6 joins the network, they accept the most common ledger, which does not include User 1's LE. If User 6 claims

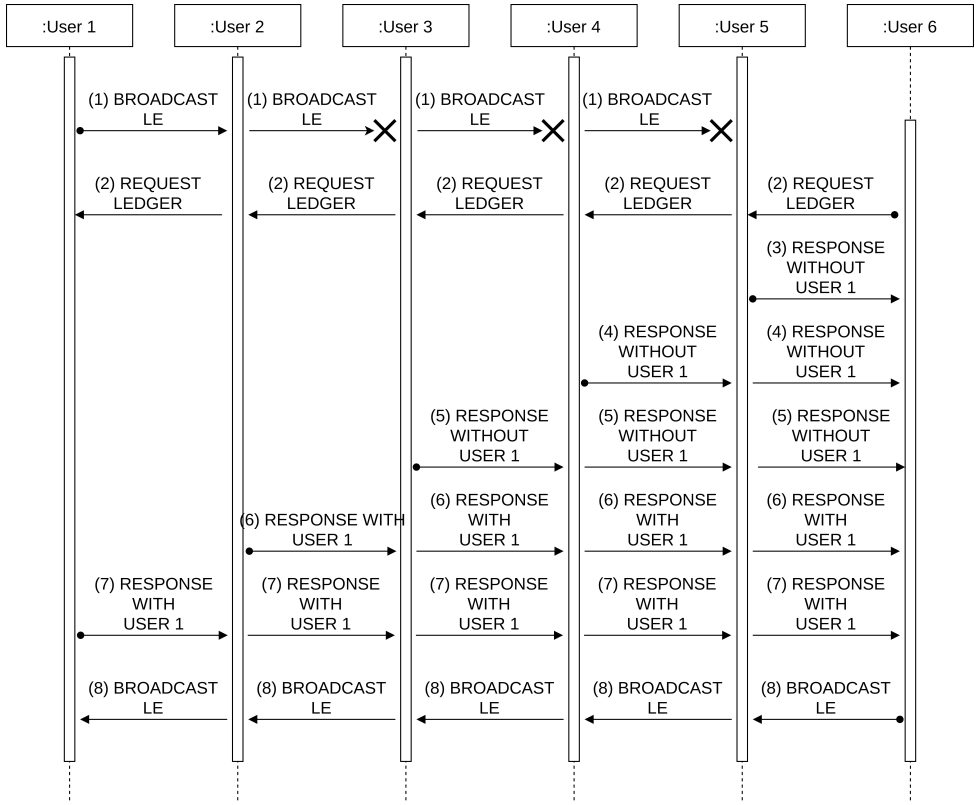


Figure 3.8: If user 6 uses the same username as user 1, user 3-5 will accept it, causing two user 1 and 2 and user 3-6 to have different versions of the ledger.

the same username as User 1, it will be accepted by User 3-5, but as long as the certificate is not CA signed, it will not be accepted by User 1 and 2. The network is now split into two groups, where User 1 and 2 have User 1's LE and the rest has User 6's LE. If a new user joins the network, everyone will participate in the voting. The ledger with User 6's LE will be chosen as the accepted ledger, and since both User 1 and User 2 can see all the votes, they will replace User 1's LE with User 6's LE. User 1 will then have to sign up again. However, should User 1 have a CA-signed certificate, User 1 and 2 will not replace User 1 in their ledger. In that case, User 1 would broadcast their LE again, and the other users would replace User 6's LE with User 1's LE, even though the majority had User 6' LE.

Conflicts such as those described above will rarely happen because the username should be an email address. If all users pick an address they own, there will be no accidental conflicts. Accidental conflicts will not happen if the majority receive the first broadcast LE neither, because the new users will then see that the username is

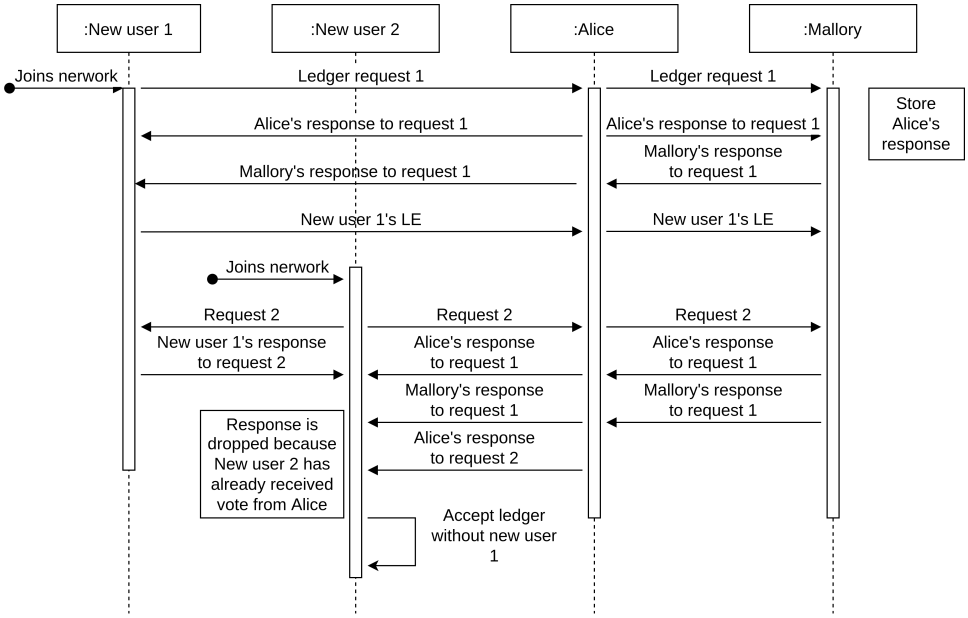


Figure 3.9: If Mallory responds to request 2 first with stored valid responses, new user 2 will accept an old version of the ledger.

taken. However, how such conflicts are handled is critical to ensure that malicious actors cannot exploit them.

3.3.7 Mitigating attacks

There are several security mechanisms added to the proposed solution to mitigate attacks. This section describes the most important ones and what attacks they mitigate.

Vote replay attack

When a new user joins the network, they first have to obtain the ledger from the existing users. To decide which ledger is correct, every user responds to the request, with the responses acting as votes. This process has been described in further detail in Chapter 3.3.3. If a valid vote in one voting could also be valid in another voting, a malicious actor could exploit this to execute a replay attack. This is shown in Figure 3.9. In the figure, Mallory stores Alice's response to request number 1. By storing the vote, Mallory can vote on Alice's behalf by replaying the stored response when a new ledger request is received. The vote would be for a previous version of the ledger and could lead to the new user accepting an outdated version of the ledger.

Nonces are introduced to avoid that the response to one request is also a valid response to any other request. The nonce is a randomly picked integer chosen by the user sending a request and is sent along with the request. The nonce has to be sufficiently large so that a user will not randomly pick a nonce that has already been used. For a response to be valid for a specific request, the response has to include the same nonce. Because an attacker cannot forge a signature, there is no way for them to obtain a valid response signed by another user, and the attack is mitigated.

Ledger rollback attack

A ledger rollback attack is illustrated in Figure 3.10. The attack is conducted by first storing the responses sent to a request. These responses will be signed and valid for a specific nonce. Because these responses act as votes, they can be used later to vote for an earlier version of the ledger. This attack is similar to the previous one, but require the attacker to send out a response with the same nonce as the stored responses. While the previous attack can be conducted on users when they try to join the network, an attacker conducting this attack must assume that all users have joined the network and know the current ledger.

In the situation shown in Figure 3.10, a new user joins the ledger after Mallory has stored responses from the other users, i.e., Cole, Bob, and Alice, with nonce 1. After the new user has joined, Mallory can send a new request with nonce 1 and quickly send the stored responses with nonce 1. Then, when Cole, Bob, and Alice send their legitimate responses in messages 10-13, the other users will discard the responses, as they have already received a signed response from them. Therefore, the stored votes will be in the majority, and all the users will accept the ledger from before New joined the network. This way, Mallory can shut New out of the ledger, making this a DoS attack.

To avoid this from happening, LEs are not removed from the ledger even when they are not a part of the accepted ledger.

DoS by request flooding

Every request for the ledger broadcast in the network triggers a response from the other users. All the users in the network then handle these responses. Flooding the network with requests will therefore cause an increase in computational load on the devices. If the computational load is sufficiently large, the device will no longer be able to handle all the messages it receives, and new users will be unable to join the network. When the computational load gets too large, the application will crash, and the service becomes unavailable.

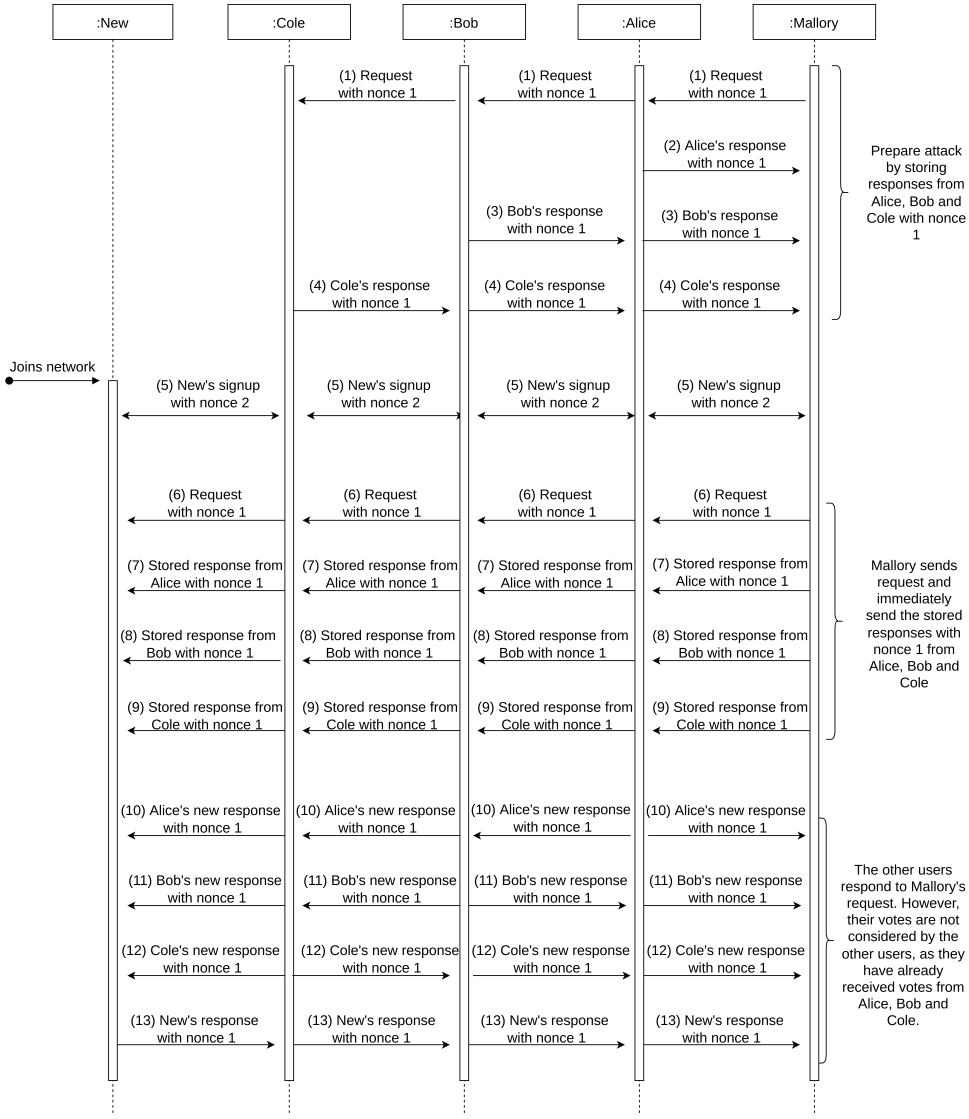


Figure 3.10: By storing responses from Alice, Bob and Cole, Mallory can force them to accept a ledger without New.

To combat DoS attack by request flooding, a user will not respond to requests for γ seconds after responding to a request for the ledger. Requests for ledgers received within this γ second window will be dropped. What value should be chosen for γ is discussed in Chapter 5.1.8.

The messages are transmitted multiple times to avoid packet loss. As long as

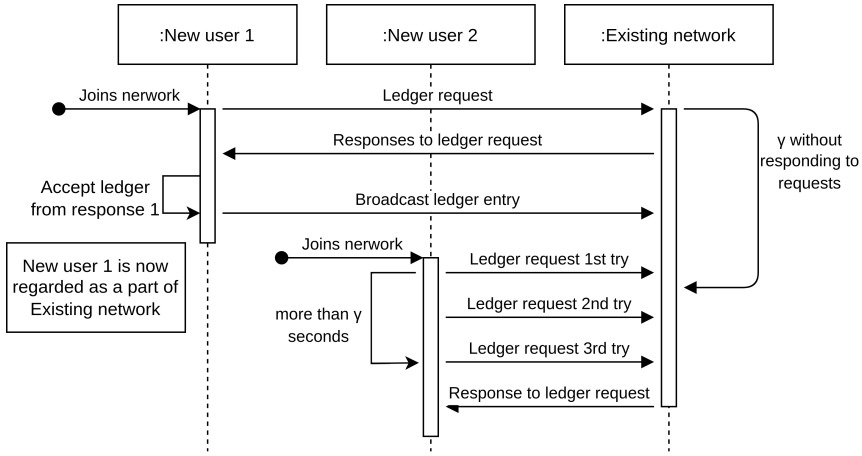


Figure 3.11: If one ledger request falls within the time window where a request is dropped, another will fall outside it.

γ is less than the time between the first and last request transmission, dropping these packets will not lead to requests not getting a response. Figure 3.11 shows why the mitigation mechanism will not lead to new users not getting the ledger. Suppose one of the transmissions is received inside the window where requests are being dropped. The system is designed to receive another one outside of the dropping window. Dropping some packets reduces the redundancy of the packets sent, hence, increasing the probability of packet loss.

3.4 Connection establishment

The first time two peers establish a connection, they set up an mTLS connection. mTLS provides confidentiality, integrity, and authentication between two parties communicating over a reliable, in-order data stream. mTLS require both parties to present a certificate, resulting in mutual authentication of the parties. Hence, the parties can trust that they are talking to their alleged peer to the same degree as they trust the certificates. The users only trust certificates already in the ledger, and the certificates have to be either self-signed or signed by the CA. To ensure message integrity in the handshake messages, the hash algorithm SHA256 is used, which is considered secure [Dan12]. As discussed in Chapter 2.1.3, the initial TLS handshake requires multiple messages to be sent and, hence, has more overhead. Therefore, an effort has been made to reduce the connection establishment time the second time two peers communicate.

After the initial mTLS handshake, the peers negotiate a symmetric key. The second time two peers communicate, they do not set up a full mTLS connection but

use pure TCP instead. The symmetric key is used to encrypt the data sent over the pure TCP connection. Every time they communicate, the peers negotiate a key they will use the next time they communicate.

The proposed solution uses DH key exchange to achieve a secure key exchange. The DH key exchange is a well-known and tested algorithm that is hard to break [MW00]. However, the DH key exchange requires both users to send their public key. By introducing DH-ratchet, two messages are only needed the first time two users communicate, while one message is required from there on.

The first DH key exchange between two users initiates a DH-ratchet. The ratchet uses one user's private key and the other user's public key and thus requires the exchange of two messages. These keys are used to calculate a shared DH secret, from which the users derive the symmetric key. From the second time two users interact, only one of the users has to update their key pair to generate a new symmetric key. Only one message is therefore required to update the symmetric key. The users renegotiate the symmetric key every time they set up a connection, hence, the system provides both forward and backward secrecy between connections.

If an attacker can break one of the symmetric keys, he will be able to read all messages within that conversation. By introducing double ratchets, this problem is reduced to a backward secrecy problem, meaning the attacker will only be able to read messages from that point in the conversation and forward in time. Double ratchets use a KDF function on every key after use to ensure a key is only used once. With double ratchets, it is not possible to find an old key given a new one, but it is possible to find a new key given an old one. Using double ratchets and DH-ratchets results in a system that provides forward secrecy within a conversation and forward- and backward secrecy between conversations [MP16a].

To encrypt and sign messages with the symmetric keys, AES in GCM is used. This algorithm is an authenticated encryption algorithm providing both confidentiality and integrity. The algorithm achieves security equal to its key size. The proposed solution uses AES in GCM with a key size of 256-bits as recommended by Android [Goo22a].

Chapter 4
Proof of concept application

This chapter describes the proof-of-concept application developed to validate the proposed solution. Specifically, the chapter includes specifications on implementation choices and technologies used to achieve the functionality of the proposed solution.

4.1 Overview

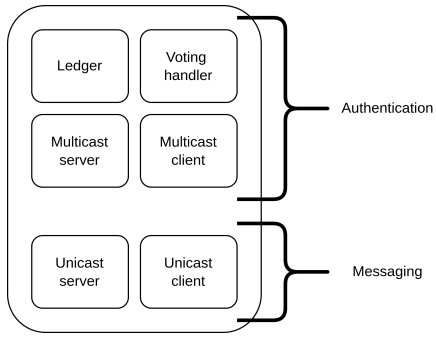


Figure 4.1: Application overview

The proof of concept application consists of six components. These are shown in Figure 4.1. In order to achieve user authentication, a multicast client, multicast server, a voting handler, and a ledger are involved. The multicast client and server respectively send and receive multicast messages. The voting handler handles processes related to consensus and sign-up. The ledger module records users associated with the network and their authentication material. The unicast client and server handle instant messaging.

An Android Service starts working as a multicast server when the application starts. The multicast server opens a multicast socket and joins a predefined multicast group. The multicast server handles all packets sent to the multicast group. Some messages are handled entirely in the multicast server, while other messages are passed

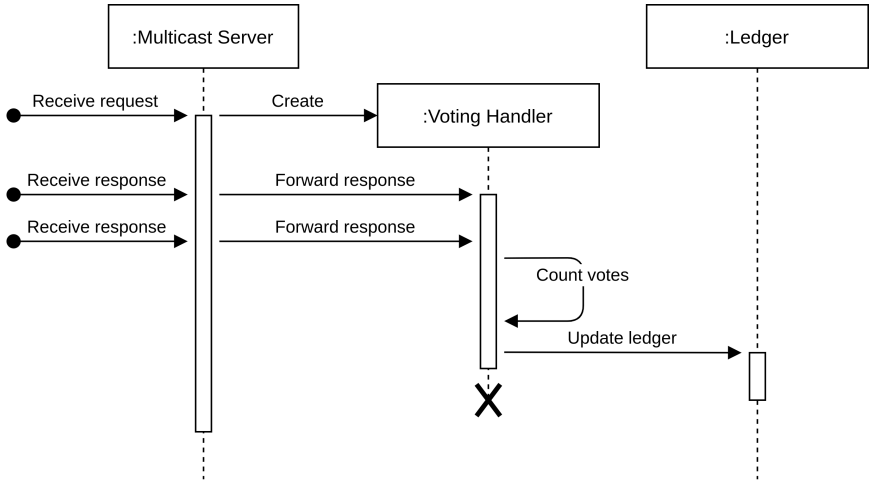


Figure 4.2: Show how the modules of the application work to handle requests.

further into the system. Chapter 4.2.1 further describes how the application handles messages.

The application initiates a multicast client along with the multicast server when it launches. The multicast client joins the same predefined multicast group and is responsible for sending messages to the multicast group. All multicast messages are sent using UDP.

Figure 4.2 shows when the application initiates a voting handler and how a ledger is updated. When a new user joins the network, the user sends a request ledger message which starts a new voting. For every new voting, the application initiates a new voting handler. The voting handler handles votes, including full ledgers and hashes, counts votes, selects the correct ledger, and is responsible for updating the ledger after a finished voting.

The ledger module contains the ledger itself and methods for updating the ledger and creating new ledger blocks. The application updates the information in the ledger after every voting.

When the application starts, it initiates a unicast server in addition to the multicast server. The unicast server listens for incoming requests to set up unicast communication. On an incoming request, the unicast server sets up a TLS over TCP or a pure TCP connection depending on the available encryption material.

When the user starts a chat, the application initiates a unicast client. The unicast client sends a connection request to its peer's unicast server and establishes

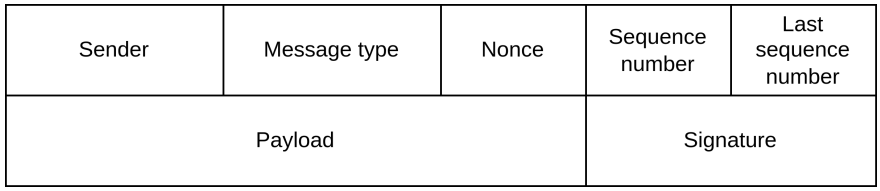


Figure 4.3: Multicast message fields

a connection.

After a unicast connection is established, both server and client send and receive messages. The types of messages is further discussed in Chapter 4.3.1.

4.2 Authentication

Consensus on the users' identity is needed to enable authentication in the application. How this is used for authentication is described in Chapter 3. This section describes the messages and modules implemented to achieve the consensus.

4.2.1 Multicast messaging

The application uses a set of predefined messages to share and synchronize the ledger between devices. These messages are sent to the predefined multicast group. Every device joins this multicast group when starting the application. Hence, a message sent in this multicast group is a broadcast in the context of the application. The following paragraphs describe the structure of multicast messages and how the different types of multicast messages are used.

As shown in Figure 4.3, a multicast message has the following fields: a sender, a payload, a signature, a nonce, a sequence number, a last sequence number, and a message type. Different message types use a different subset of these fields. The purpose of each field and its usage in the different message types are further described below.

The sender field holds the username or LE of the sender. The receiver uses this information to look up authentication material for signature verification.

The payload field holds the information carried with the message. The different message types use different but standardized formats for the payload. It typically includes a ledger, a ledger hash, or a username.

The signature is made with SHA256 Elliptic Curve Digital Signature Algorithm (ECDSA) and uses the payload and nonce as input. The signature helps provide

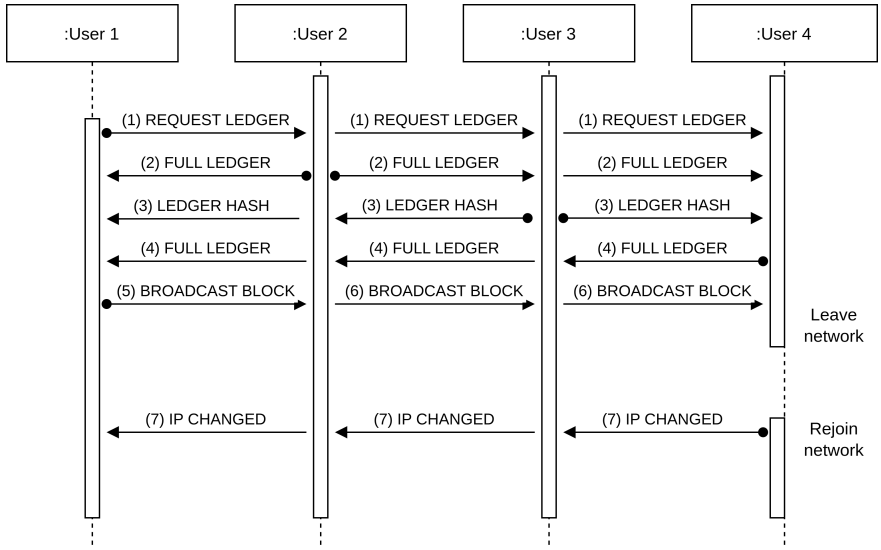


Figure 4.4: Multicast message exchange

message integrity.

The nonce identifies to which voting process the message is related. Before the application sends a request ledger message, a nonce is generated and appended to the request. This nonce will serve as the identity of the voting process. All responses related to this voting process will use the same nonce. The use of nonce also helps mitigate the replay attack described in Chapter 3.9.

The sequence number and last sequence number are used when packets are fragmented at the application layer. Because the messages are sent over an unreliable UDP channel, there is a probability of messages getting lost. Fragmenting big messages reduces the probability of a lost message. How fragmentation reduces packet loss is further discussed in Chapter 5.1. The sequence number and last sequence number fields aid in packet reassembly. How messages are fragmented is further described when discussing FULL_LEDGER message below.

The message type indicates how the receiver should handle the packet. Five different multicast message types are defined. These are discussed below. Figure 4.4 and Figure 4.5 shows how the different message types are typically triggered.

Multicast message types

REQUEST_LEDGER is the first message sent when a user starts the application. The message starts a new voting process where the intention is for the new user to

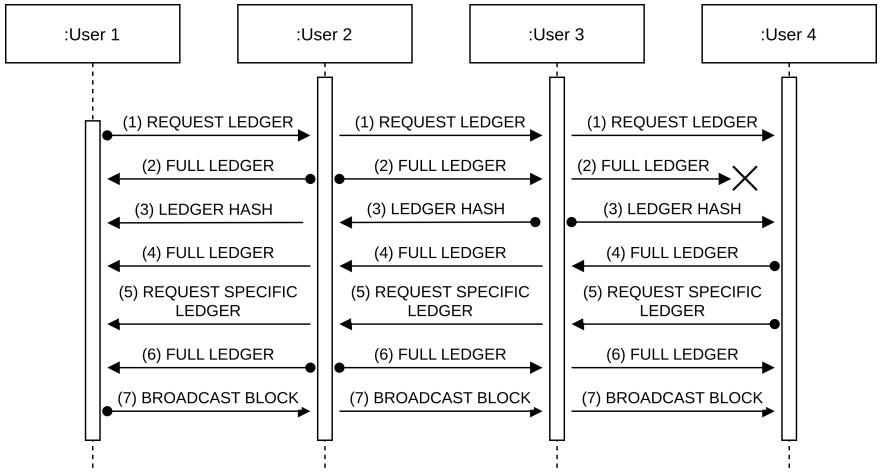


Figure 4.5: Multicast message exchange with user 4 not receiving full ledger

learn about its peers by receiving the full correct ledger. This message type uses the message type field and nonce field. The nonce field identifies the voting process this message initiate. All messages that are related to this voting will use the same nonce. The use of nonce as identification for voting allows multiple users to join the network simultaneously as multiple votings can proceed in parallel.

REQUEST_SPECIFIC_LEDGER is used to request the ledger from a specific user. If a voting is finished, but only hashes of the most common ledger have been received, the user sends this message to get the full ledger. This scenario is illustrated in Figure 4.5 where message number 2 fail to reach User 4, and User 4 only receives the hash of the ledger. The REQUEST_SPECIFIC_LEDGER message uses the payload, message type, and nonce fields. The payload consists of the username of the peer that should respond and a hash of the requested ledger. The user who should respond is selected randomly from those who responded with the most common hash. Because the REQUEST_SPECIFIC_LEDGER message is broadcast, the username is needed in the payload to indicate which user should respond to the request. As the REQUEST_SPECIFIC_LEDGER message and its response are broadcast, all users can listen to the request and its responses. This means only one user has to send this message which reduces the load on the network. The hash in the payload field allows the respondent to ensure the correct ledger is sent in the response.

The application uses the FULL_LEDGER message as a response to REQUEST_-LEDGER or REQUEST_SPECIFIC_LEDGER. This message type uses all the fields of the multicast message. This message may be fragmented to support sending of large ledgers. The FULL_LEDGER message is fragmented if the ledger has more than four entries. This is further discussed in Chapter 5.1. In the first fragment,

the sender field holds the LE of the sender. The LE includes the sender's certificate, which allows the receiver to verify the message signature. In the following fragments, the sender field holds the sender's username. This reduces the packet size while still allowing the receiver to look up previously received encryption material. If any other fragments are received before the first fragment, their integrity cannot be verified as only the first fragment include the public key needed to verify the signature. The fragments are then stored temporarily until the first fragment is received. When the first fragment is received, the other fragments can be verified. The sequence number and last sequence number indicate which number in line a fragment is and how many fragments there are. This information is used when reassembling the ledger. The signature provides message integrity and ensures every user only can contribute with one vote per voting.

LEDGER_HASH is used as a response to REQUEST_LEDGER and contains a hash of the full ledger. This message type uses the sender, message type, payload, signature, and nonce fields. The sender field holds the LE of the sender. The payload contains a hash of the ledger. As with the FULL_LEDGER message, the signature provides message integrity and is used to ensure every user only contributes with one vote per voting. The nonce value is the same as the nonce in the REQUEST_LEDGER message, which triggered this response.

BROADCAST_BLOCK is used for broadcasting an LE in the network. After completing a voting and receiving the correct ledger, a new user sends this message to broadcast their LE to the network. This message type uses the sender, message type, payload, signature, and nonce fields. The sender field holds the username of the sender. The payload holds the LE of the new user. The signature proves that the new user knows the private key related to the public key of the LE. The nonce value is the same as the nonce in the REQUEST_LEDGER message, which triggered this response.

IP_CHANGED is used to notify the network when a user changes IP address. Message 7 in Figure 4.4 illustrates the use of this message type, where a user leaves the network, then returns later with a new IP address. This message type uses the sender, payload, signature, and message type fields. The sender field holds the ledger entry of the sender. The payload holds the new IP address and the time of updating the IP address. Including the time in the payload combats the replay attack. Without this measure, an attacker could record the IP_CHANGED message and replay it later, performing a DoS attack. Finally, the signature provides message integrity and ensures only the owner of the LE can send a valid IP_CHANGE message.

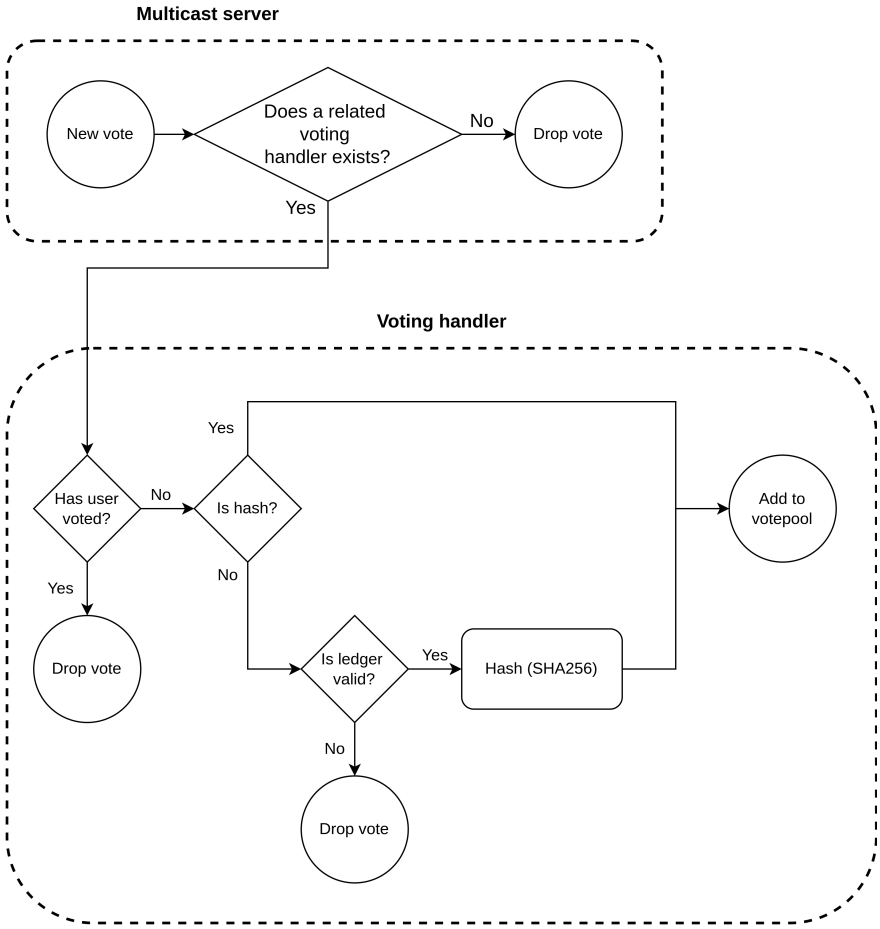


Figure 4.6: Voting handler flowchart

4.2.2 Authentication modules

The voting is an essential process used to achieve ledger consensus. The voting handler handles messages and processes related to a voting. It does so by storing the votes received, both full ledgers and hashes, counting votes, and selecting the correct ledger after a voting is finished. It is also responsible for updating the ledger after it learns the correct one. The voting handler is initiated when a REQUEST_LEDGER message is sent or received. The nonce from the REQUEST_LEDGER message identifies the voting handler. A new voting handler is initiated with every new voting.

Figure 4.6 shows how a new vote is processed when entering the system. First, the multicast server receives and interprets the message received. If the received message is of type FULL_LEDGER or LEDGER_HASH, the message is interpreted

as a vote, and the multicast server checks the nonce to determine if a related voting handler exists. If this is the case, the multicast server forwards the message to this voting handler. If no related voting handler exists, the message is dropped.

Every user can only vote once. Hence, when a voting handler receives a new vote, the certificate of the incoming vote is compared to the certificates of the previously received votes to check if the user has already voted. If no match exists in this comparison, the vote is accepted. If a match exists the vote is dropped. The certificates are public, and a malicious user could easily send a vote with another user's certificate to vote on behalf of another user. Therefore, the voting messages are signed, and votes with an invalid signature are dropped. This results in a system where the voter has to know a certificate's corresponding private key to vote with the certificate.

If the received vote is from a user who has not already voted, the received voting message is processed further by being formatted into a vote. If the voting message is a LEDGER_HASH message, it is in the correct format and is added to the voting pool together with the LE of the sender. If the voting message is a FULL_LEDGER message, it must be validated and hashed before sending it to the voting pool. The validation is done by iterating over the LEs to ensure no username duplicates exist. If username duplicates exist, the ledger is considered corrupt, and the voting handler drops the vote. If the ledger is valid, it is hashed with the SHA256 and added to the voting pool together with the LE of the sender.

The voting handler will end a voting and choose a ledger after one of the criteria described in Chapter 3.3.5 are fulfilled. The first criteria will accept a ledger if there are at least two votes from CA-certified users for the same ledger, and they make up more than 50% of the total number of votes. However, to fulfill the remaining two criteria, a timer is needed. When a new user joins the network, the user does not know how many users are active in the network, and hence, the new user can not know how many votes to expect in a voting. The acceptance timer ensures that the voting process always finishes and a ledger is accepted. Figure 4.7 shows that the acceptance timer starts when the voting handler is initiated. The timer is set to α seconds. The value of α must ensure that the user receives all voting messages within this period. The value of α is further discussed in Chapter 5.1.5. When the timer finishes, the ledger is accepted according to the last two criteria described in Chapter 3.3.5.

In addition to the acceptance timer, a timer to check if the user is alone in the network starts when the voting handler is initiated. The timers are shown in Figure 4.8. The alone-in-network timer is set to β seconds. The value of β must ensure that potential voting messages reach the user before the timer finishes. The value of β

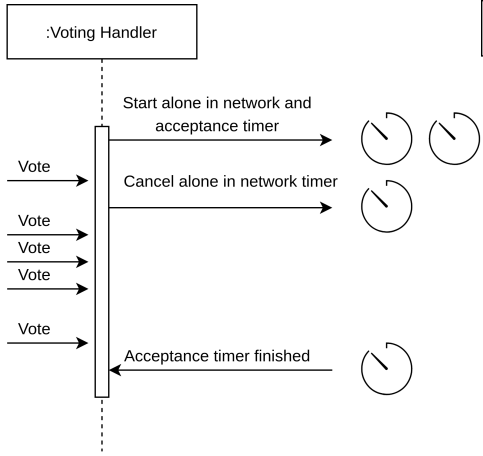


Figure 4.7: Acceptance timer

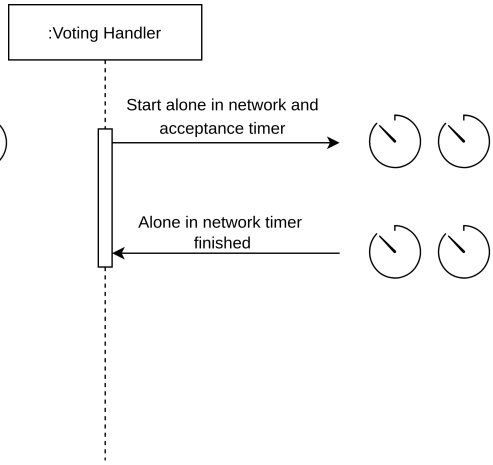


Figure 4.8: Alone in network timer

is further discussed in Chapter 5.1.6. If the alone-in-network timer finishes before receiving any votes, the system assumes the new user is the first in the network, and a new ledger is initiated.

4.2.3 Certificates

The application generates a key pair and a certificate when the user signs up for the application. The certificate is generated using Bouncycastles X509 v3 Certificate Builder and is signed using Bouncycastles ECC Content signer [The21]. If the user signs up when offline, the certificate is generated on the user's device and is self-signed. If the user signs up online, the user will generate a key pair and send the public key and username to the CA server running on the Internet. The CA server is a Google Cloud function in the proposed solution. The CA server responds with a CA-signed certificate. Both the CA- and self-signed certificate has a validity of 1 year. All users have access to the CA server's public key and can verify CA-signed certificates.

For a user to receive a CA-signed certificate, the user will have to sign up online and verify their email. Online authentication and log-in are handled with Firebase [Goo22b]. In addition, Firebase is responsible for checking global username availability, password strength, and handling email verification.

Certificates are stored on the devices using Android Keystore [And22]. The user's own certificate is stored in a Keystore instance of the Android Keystore, while the peer certificates are stored in a Truststore instance of the Android Keystore. After a successful certificate generation, the certificate is stored in the Keystore as a root certificate. This certificate will be shared with peers when joining the ledger and

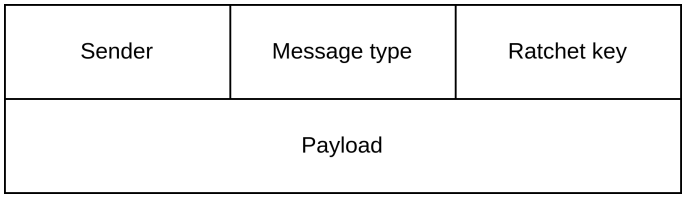


Figure 4.9: Unicast message fields

during the TLS handshake. When a user is added to the ledger, they broadcast an LE that includes a certificate. When their peers receive this LE they will add the certificate to their Truststore. On an incoming TLS connection request, the application looks for certificates in the Truststore. If a valid certificate exists for the user requesting a connection, the user is regarded as trusted, and they can establish a connection.

4.3 Messaging

The service provided by the proof of concept application is an instant messaging service. This chapter describes the messages and modules needed to provide this service.

4.3.1 Unicast messaging

A set of predefined messages is used to establish a connection and send messages between devices. These messages are unicast and are sent over a reliable mTLS over TCP or pure TCP connection. When the application uses the different technologies is further described in Chapter 4.3.2.

The unicast messages in the proof of concept application are encrypted at the application layer using AES in GCM. As described in Chapter 2.1.3, AES in GCM provides both confidentiality and integrity of the data. Hence, together with the ledgers authenticity property, this type of encryption provides integrity on the application layer.

As shown in Figure 4.9, a unicast message has the following fields: a sender, a payload, a ratchet key, and a message type. The different message types use a different subset of these fields. The purpose of the fields and how the different message types use these fields are further described below.

The sender field holds the username of the sender. The receiver uses this information to look up authentication and encryption material.

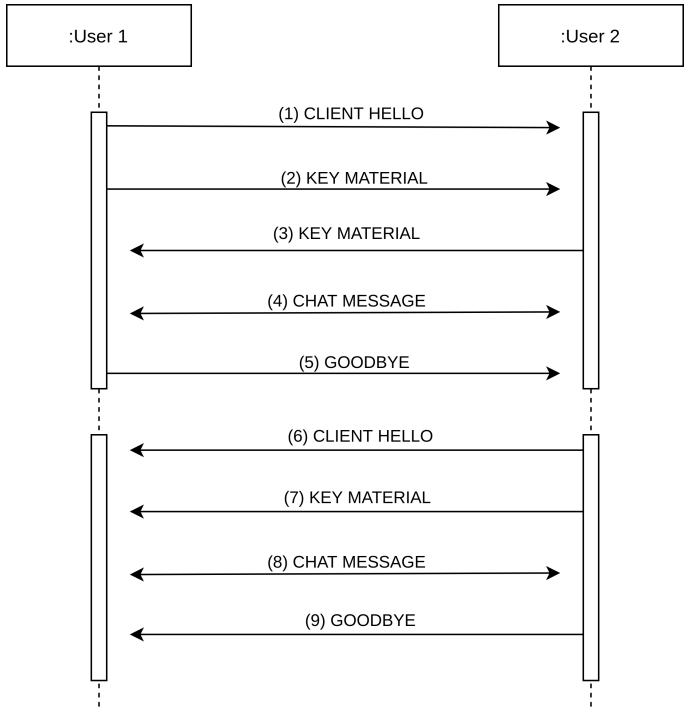


Figure 4.10: Unicast message exchange

The payload field holds the information sent in the message. This information is typically the content of the chat message the users send.

The ratchet key field holds information on the current ratchet key round represented as an integer. If the ratchet keys get out of sync, this integer indicates how many times the original key has been hashed to produce the current symmetric encryption key.

The message type indicates how the receiver should handle the packet. Four different unicast message types are defined. These are discussed below. Figure 4.10 show how the different message types typically are triggered.

Unicast message types

CLIENT_HELLO is sent as the first message in a message exchange and initiates the chat. This message type uses the sender, the message type, and the ratchet key field. Upon receiving a CLIENT_HELLO message, the receiver opens a new server socket used to send and receive messages in the current conversation. This server socket is active until the server receives a GOODBYE message from the same peer.

When the client sends the CLIENT_HELLO message, the system can not guarantee a negotiated encryption key. Hence, the CLIENT_HELLO message is not encrypted.

GOODBYE is sent when the client device leaves the chat. This message type uses the sender, the message type, and the ratchet key field. Upon receiving a GOODBYE message, the receiver closes the related server socket. The GOODBYE message is encrypted with the symmetric key, ensuring that a malicious actor can not send a GOODBYE message to trigger a socket close.

CHAT_MESSAGE is sent to transfer chat data. This message type uses all the message fields. The payload field holds the chat message to be transferred. This message type is encrypted with the symmetric key to provide confidentiality and integrity.

KEY_MATERIAL is sent to transfer keying material. This message type uses all the message fields. The payload field holds the key material needed to do a DH key exchange. This message type is encrypted to provide confidentiality and integrity. As Figure 4.10 shows, the first time two users communicate, both users have to send this message in order to agree on a symmetric key. The second time two users communicate, the DH-ratchet provides the mechanisms to allow only one of the users to send the key material. DH-ratchets are discussed in Chapter 2.1.3.

4.3.2 Messaging modules

In order to provide the messaging service, the application uses two modules: a unicast client and a unicast server. The unicast client is initiated when the user opens the chat activity. The user can only be in one chat activity at a time, meaning only one unicast client is running at a time. The unicast client handles symmetric encryption material and encrypts and sends unicast messages. The application terminates the unicast client when the user leaves the chat activity.

When establishing a new connection, the unicast client decides if the connection is to be an mTLS over TCP or a pure TCP connection. If two users have not been in contact with each other before, no symmetric encryption material exists for these users, and an mTLS over TCP connection is established. If the users previously have been in contact, a symmetric key has been negotiated. This key will be used for encryption, and the additional mTLS encryption is not needed. Hence a pure TCP connection will be established from the second time two users communicate. Not needing to establish a mTLS connection reduces the connection setup time. The benefits of establishing a pure TCP connection is further discussed in Chapter 5.2.2.

The application initiates a new unicast server when the user receives a CLIENT_-HELLO message. As a result, multiple unicast servers can run simultaneously,

allowing the application to support receiving messages from multiple devices simultaneously. The unicast server is terminated when the user receives a GOODBYE message.

The unicast client and unicast server maintain two ratchets for sending unicast messages. The sending ratchet maintains the symmetric key used for encrypting messages, while the receiving ratchet maintains the symmetric key used for decrypting messages. Together they ensure that every encryption key is only used once [MP16a]. Ratchets are discussed in Chapter 2.1.3.

4.4 Activities

Android divides applications into different interfaces called activities. The proof-of-concept application consists of four activities: The main activity, the chat activity, the sign-up activity, and the login activity.

4.4.1 Main activity

The main activity is the first interface presented to the user. This activity is responsible for initiating the Keystore and Truststore and starting the multicast and unicast servers.

Figure 4.11 shows the main activity interface. The interface displays the content of the ledger as a list of users. Each list entry has a username, an IP address, a hash of the certificate, and an indication of the certificate's type. A yellow dot indicates a self-signed certificate, and a green dot indicates a CA-signed certificate. If the user taps a list entry, the application initiates a chat with that user.

4.4.2 Chat activity

When a connection between two peers has been initiated, the chat activity starts. The chat activity is shown in Figure 4.12. In this activity, the users can read and send messages. Messages are displayed in chronological order, with outgoing messages on the right-hand side and received messages on the left-hand side. The user writes their message in the bottom edit text and sends the message by pressing the send button.

4.4.3 Other activities

If a user wants to sign up for the application, the user will start the sign-up activity, shown in Figure 4.13. The sign-up activity displays fields for email and password and a button for signing up. The user must fill in these fields with a valid email

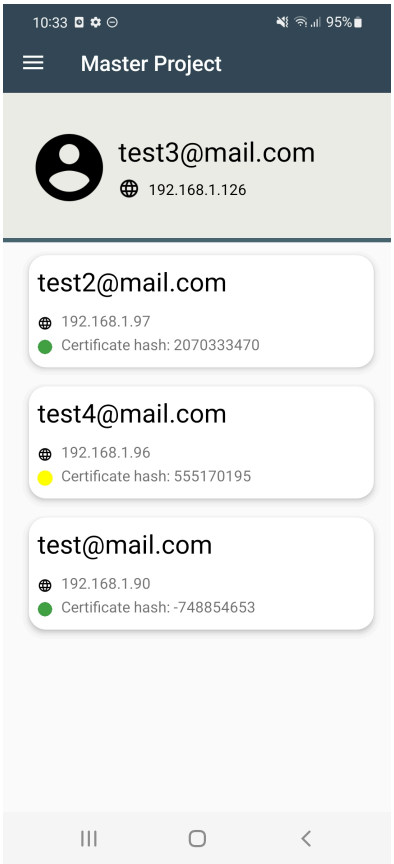


Figure 4.11: Main Activity

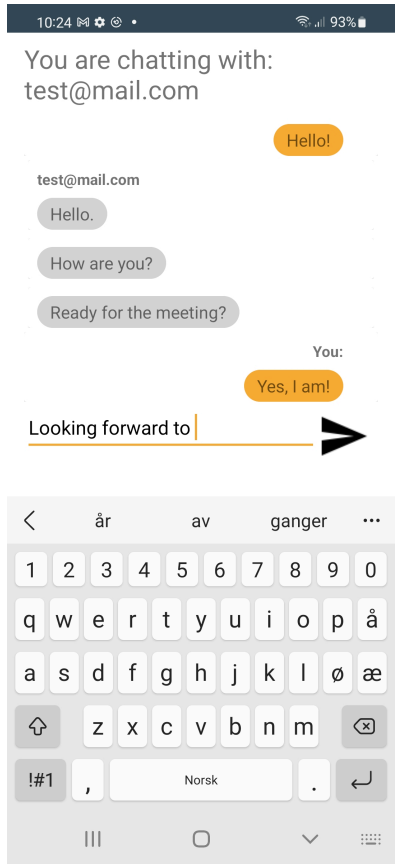


Figure 4.12: Chat Activity

and password to sign up. If the user is online, a user will be created in Firebase. Regardless of the Internet availability, an LE will be created and broadcast.

If a user is online and has already created a user in Firebase, they can use the login activity shown in Figure 4.14. First, the user has to enter their credentials, and if the credentials are correct and they have validated their email, they will receive a CA-signed certificate.

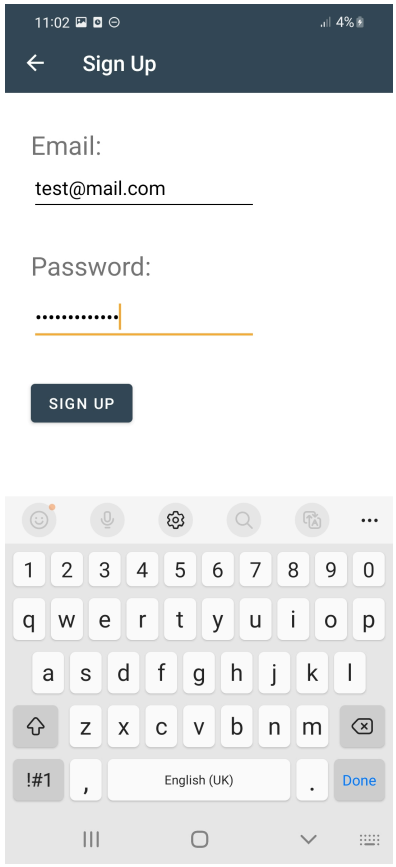


Figure 4.13: Sign Up Activity

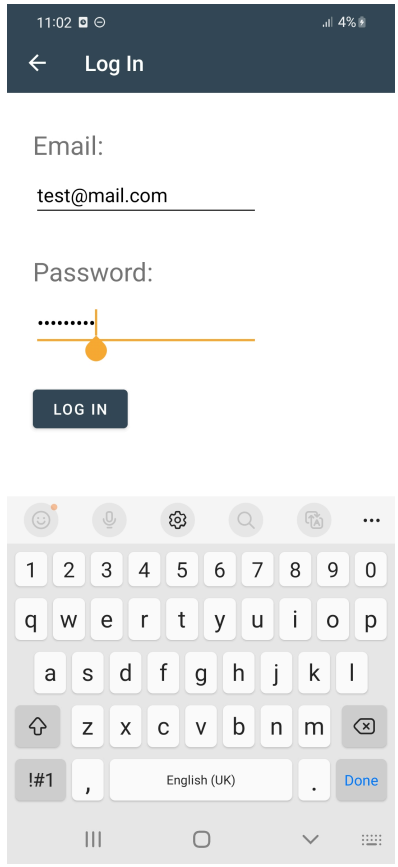


Figure 4.14: Log In Activity

Chapter 5
Results

This chapter describes the test conducted to verify the security and performance of the proof of concept application. The chapter also presents and discusses the results and recommends optimal parameters to achieve the best security and performance.

5.1 Optimizing parameters

The application uses UDP as transport layer protocol for multicast messaging, used for ledger management. UDP is a best-effort protocol which means packets may be lost. The loss of multicast packets will affect the system's ability to synchronize the ledger and achieve a consensus on users' identities. As described in Chapter 3.1.2 it is essential for the system's authentication mechanism to achieve consensus. Thus, the system's ability to deliver multicast packets is essential for performance and security. To reduce the packet loss and optimize the performance of the solution, there have been conducted tests to see how different parameters affect the system's performance. In this section, the test results used for deciding the value of those parameters are presented, and a recommendation for each parameter is provided.

In the tests, one user is active in the network as a second user joins the network. When the second user joins the network, they will request the ledger from the first user, which will send a pre-programmed ledger to the multicast group. The tests only regard packet loss, so there is no need for multiple peers to vote for a correct ledger. The messages that are sent and received are logged. Based on these logs, it is possible to determine whether the user has received the entire ledger. Each ledger is seen as one packet in the tests, although it is transmitted in fragments. Therefore, a ledger is considered lost if one or more fragments are not received. The tests have been conducted using two Samsung Galaxy S21 5G phones and Netgear a MR2100 Nighthawk M2 wireless router.

There are several parameters that may influence system's performance and the main ones have been identified as:

- The number of transmissions of each packet
- The time between transmissions
- The fragment size
- The time between fragments

The number of parameters makes it time consuming to test all the different combinations of possible values. Due to the time limitations of this project, the optimal value of each parameter has been found by varying the value of the parameter in question while keeping the values of the other parameters fixed. The fixed parameters' values are chosen to be near-optimal, based on a small sample of test results.

The number of LEs in the pre-programmed ledger in the tests negatively affects the packet loss. For each test, the number of LEs is chosen so that the packet loss will be near to 50% for what is believed to be the optimal value for the tested parameter. A packet loss of around 50% will increase the statistical significance of the results compared to a very high or very low packet loss. The test environment for testing the parameters identified above are deliberately constructed to have a large packet loss for the purpose of increasing the statistical significance. It is designed solely to find the optimal values for the parameters tested. Hence, the results do not represent how the system will generally perform under normal operation. How the system will perform under normal circumstances is illustrated further in Chapter 5.2.

When calculating the packet loss, a full ledger is considered one packet, even though it is sent in multiple fragments. The reason is that without all the fragments the hash of the full ledger cannot be computed and it cannot be counted as a vote. The packet is considered successfully received if all fragments are received at least once. That is because the entire ledger will have to be received in order for it to be considered a vote. The packet loss is the number of full ledgers received divided by the number of full ledgers sent.

5.1.1 Number of transmissions

Multicast packets are sent multiple times to address the challenge of the unreliable UDP protocol. By sending the packets multiple times, the chance of losing the message is reduced. However, sending the packets too many times will be an unnecessary load on both the devices and the network. Hence a test is conducted to find the optimal number of transmissions needed.

Figure 5.1 shows how the packet loss is affected by the number of transmissions of the same packets. The orange bars show the packet loss for the REQUEST_LEDGER message, while the blue bars show the packet loss for the FULL_LEDGER message

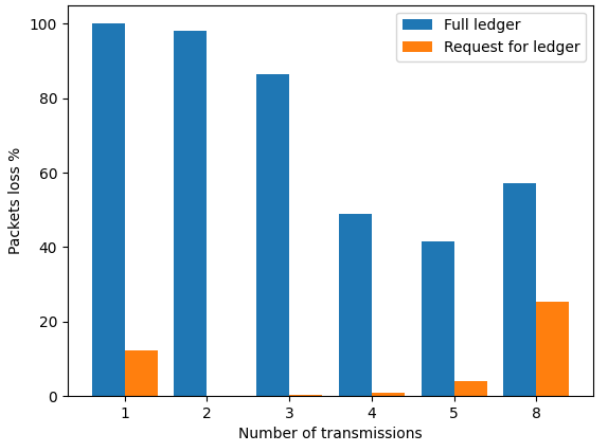


Figure 5.1: Number of transmissions

Number of transmissions	Number of trials	Packet loss	Confidence interval 95%
1	300	12.3%	[8.834, 16.597]
2	300	0.0%	[0.000, 1.222]
3	300	0.3%	[0.008, 1.843]
4	300	1.0%	[0.207, 2.894]
5	300	4.0%	[2.084, 6.883]
8	300	25.3%	[20.510, 30.651]

Table 5.1: Statistical values for successful packet delivery given number of transmission of same REQUEST_LEDGER packet

Number of transmissions	Number of trials	Packet loss	Confidence interval 95%
1	300	100%	[98.779, 100.000]
2	300	98.0%	[95.698, 99.263]
3	300	86.3%	[81.920, 90.011]
4	300	48.8%	[42.882, 54.478]
5	300	41.7%	[36.028, 47.472]
8	300	57.1%	[51.185, 62.675]

Table 5.2: Statistical values for successful packet delivery given number of transmission of same FULL_LEDGER packet containing 100 LEs

sent in response to those requests. The packets with requests for ledgers are sent in one fragment, and the packet loss will therefore be similar for other packets sent in one fragment, such as messages containing the hash of the ledger. In contrast, the full ledger contains 100 LEs in this test, and with a fragmentation size of 1, they are sent in 100 fragments. With more fragments, the probability of losing one fragment increases. Hence, the packet loss of the full ledger is expected to be higher than that of the requests.

As most packets are sent in only one fragment, the packet loss shown in the orange bars represents the packet loss in most packets. Hence, it is important to keep this packet loss sufficiently low. However, the packet loss of the full ledger dictates how many times the ledger will have to be requested and thus how long it takes to sign up. Therefore, when choosing the number of transmissions, the aim is to find a value that gives a low packet loss for messages of one fragment while still giving a low packet loss for the full ledger. The value that is considered optimal is, therefore 4 transmissions.

5.1.2 Time between transmissions

Another measure to reduce packet loss is to stretch out the transmissions in time. Because CSMA/CA and UDP are unreliable, packets can be lost for reasons such as collision, interference, and device congestion. Stretching the transmissions out in time reduces the probability that the same factor will affect multiple transmissions- However, too long time between packets will increase the time it takes to sign up, potentially affecting the user experience.

The tested parameter is the time between transmissions. This parameter describes the time from sending the last fragment of the ledger in one transmission until sending the first fragment of the ledger in the subsequent transmission.

Figure 5.2 shows how the packet loss is affected by the time between transmissions. The figure shows that increasing the time between transmissions will decrease the packet loss until 400 ms. The reason for this can be that what causes packets to be lost in one transmission might be gone before the next one. From 400 ms and upwards, the packet loss increases with more time between transmissions. This indicates that the advantage of stretching out the transmissions in time is reduced for values larger than 400 ms.

The number of transmissions has to be considered when analyzing the test results. While 400 ms is the optimal time between transmissions for four transmissions, according to these results, that might not be the case for another number of transmissions. That is because 400 ms between each transmission will result in 1200 ms from the first transmission starts until the last transmission starts. With four

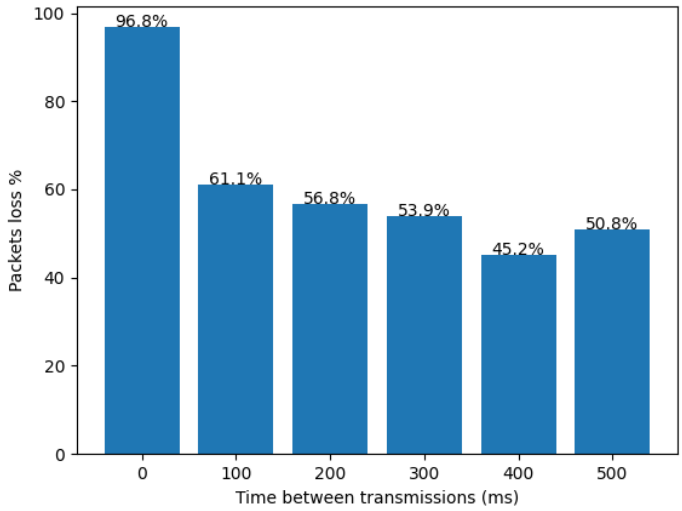


Figure 5.2: Time between transmissions

Time between transmissions	Number of trials	Packet loss	Confidence interval 95%
0ms	300	96.8%	[93.955, 98.390]
100ms	300	61.1%	[55.227, 66.553]
200ms	300	56.8%	[50.850, 62.350]
300ms	300	53.9%	[48.178, 59.742]
400ms	300	45.2%	[39.278, 50.822]
500ms	300	50.8%	[45.763, 55.560]

Table 5.3: Statistical values for successful packet delivery given time between transmissions

transmissions, 400 ms between each transmission will stretch the transmission out to 1200 ms in total. The total stretch in time will be lower for fewer transmissions, and the optimal value for the time between transmissions might be higher.

If another value for the number of transmissions where selected, the time between transmissions should be tested again to ensure that the value chosen is indeed optimal.

5.1.3 Ledger fragment size

When the ledger is sufficiently large, it is divided and sent in different fragments, as described in Chapter 4.2.1. That reduces the size of each packet sent and the chance of the packet being lost. In the proof-of-concept application, one fragment holds n LEs. The number of LEs in each fragment affects the size of each UDP message sent, as well as the number of UDP messages that have to be successfully received in order for the full ledger to be correctly assembled at the receiver. For the ledger to be counted as a vote, every fragment must be received. Thus, the number of fragments will affect the probability of the ledger being received.

When the size of each fragment is reduced, the probability that one fragment is lost is also reduced as found in [KW05]. However with smaller fragments, the number of fragments that has to be successfully received also increases. Hence, when choosing the fragmentation size, a compromise between the number of fragments and fragmentation size has to be made to optimize packet delivery. To find the optimal ledger fragmentation size, a test has been conducted by varying the number of ledger entries per fragment and observing the impact it has on the packet loss.

Figure 5.3 shows that the packet loss is lowest for a fragment size of 4 and is larger for smaller or larger fragment sizes. With fragment sizes lower than 4, the large number of packets that have to be received for a successful message delivery affects the system more than the benefits of reducing each packet size. For fragment sizes larger than 4, however, the increased chance of losing a packet when the fragments are larger is more significant than the benefit of sending fewer fragments. Hence, the optimal compromise between the number of fragments and fragmentation size is achieved when each fragment contains 4 LEs.

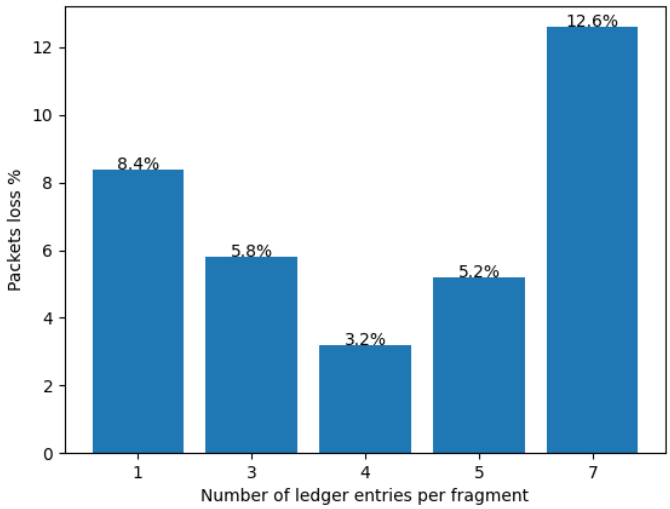


Figure 5.3: Fragment size

Fragment size	Number of trials	Packet loss	Confidence interval 95%
1 LE	500	8.4%	[6.121, 11.185]
3 LEs	500	5.8%	[3.918, 8.224]
4 LEs	500	3.2%	[1.840, 5.145]
5 LEs	500	5.2%	[3.425, 7.527]
7 LEs	500	12.6%	[9.820, 15.307]

Table 5.4: Statistical values for successful packet delivery given fragment size

5.1.4 Time between fragments

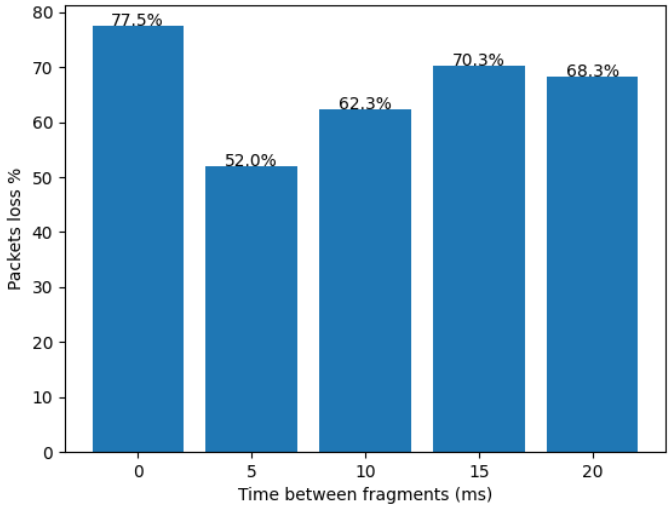


Figure 5.4: Time between fragments

Time between fragments	Number of trials	Packet loss	Confidence interval 95%
0ms	300	77.5%	[72.171, 81.947]
5ms	300	52%	[46.184, 57.776]
10ms	300	62.3%	[56.583, 67.837]
15ms	300	70.3%	[64.815, 75.445]
20ms	300	68.3%	[63.626, 72.768]

Table 5.5: Statistical values for successful packet delivery given time between fragments

When a ledger is fragmented, the time between fragments is how long the sender waits between sending one fragment and the next. The time between fragments helps stretch the transmission out in time. That will reduce the probability that multiple fragments are lost due to the same factor, as also mentioned in Chapter 5.1.2. The test has been conducted with a set of time between fragments from 0 to 20 ms.

Figure 5.4 shows how the packet loss is affected by the value of the time between fragments. As shown, packet loss decreases when increasing the time between

fragments from 0 ms to 5 ms. However, for values higher than 5 ms, the packet loss increases as the value increases. Based on these results, 5 ms has been chosen to achieve the lowest possible packet loss. Further values for the time between fragments have not been investigated as too much time between fragments will result in a system that uses a long time to send packets, affecting the user experience.

5.1.5 Time to receive full ledger

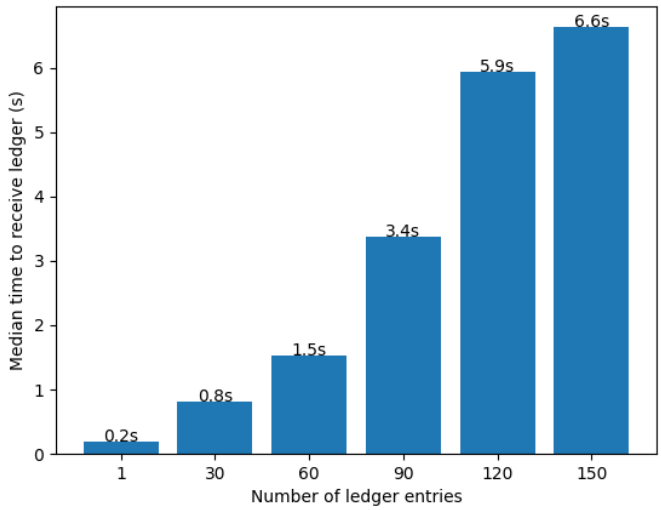


Figure 5.5: Median time to receive full ledgers of length n

Ledger size	Number of trials	Median	Variance	Confidence interval 95%
1 LE	300	0.2s	0.048211	[0.218, 0.221]
30 LEs	300	0.8s	0.676108	[0.824, 0.835]
60 LEs	300	1.5s	2.635382	[1.859, 1.887]
90 LEs	300	3.4s	6.042278	[3.283, 3.341]
120 LEs	300	5.9s	9.880437	[5.517, 5.646]
150 LEs	300	6.6s	1.471836	[6.115, 6.548]

Table 5.6: Statistical values for time to receive full ledger given ledger length

Every request for the ledger triggers a response from all the members of the ledger. However, a user cannot know how many responses they will receive and thus when all the other users have voted. That is due to packet loss, the fact that users

might disconnect from the WLAN, and that the user requesting the ledger does not already hold the ledger.

In some cases, it is possible to decide which ledger is correct before votes from all users are received. This is described in Chapter 3.3.5. However, when this is not the case, the user joining the network will have to wait until they believe they have received all votes before deciding which ledger is correct. However, a new user does not know how many users are active in the network and hence how many votes they are expected to receive. Thus, the new user waits a certain amount of time before accepting a ledger to increase the probability of receiving all votes.

If a user does not receive the full ledger that is accepted, they will request the ledger from another user. If the full ledger is still not received, the user will continue to request the full ledger over and over again until it is successfully received. To avoid unnecessary transmissions of the full ledger, the user will not request the ledger again unless the ledger is believed to be lost. From the user requests the ledger, until they send a new request for the full ledger, they will therefore wait at least the time it takes to receive a full ledger.

Figure 5.5 shows the median time it takes to receive a full ledger of length n. In reality, the users would receive not only a full ledger but also several hashes. Because all the messages are handled in the same thread on the device, many hashes would affect the time it takes to handle the responses. That would cause an increase in time before the device would consider the full ledger as received. This increase should be considered when choosing the value of the accept ledger timer.

If the value is too low, the effects will not be severe. If the ledger is requested again while in transit it will cause unnecessary traffic. If the ledger is accepted before the full ledger is received, the correct ledger will be decided with one vote missing, as the full ledger acts as a vote. This will only happen with large ledgers where it is unlikely that one vote will change the outcome, as all benign nodes will vote for the same ledger. The only consequence will therefore be an extra transmission of the ledger.

When choosing how long a user should wait to accept a ledger, it is important to consider that the user experience will be reduced if it takes too much time to accept a ledger. By setting the timer to 4000 ms, ledgers of size 90 are most likely received. For ledgers larger than 90, the packet loss is so high that the ledger most likely will have to be sent multiple times. A timer of 4000 ms is not expected to affect the user experience either, as it will only affect the time before a user can sign up or see other members in the network. The timer will most likely not affect the time it takes to sign up, as inputting user data takes more than 4000 ms. A delay of 4000 ms before a user can see the other members is also comparable to what it takes with an instant

messaging application in an online scenario.

5.1.6 Alone in network timer

New users cannot know if there are any users in the network when they join. Hence, they always have to send out a request for a ledger. If the new user does not receive any responses, they can conclude that they are the first user in the network. The new user will then create a new ledger containing their own LE. To avoid the new user creating a new ledger when there is an existing one, the new user has to wait until they can be sure that they will not receive any responses to the ledger request.

As described in Chapter 5.1.1 each packet is sent four times. The time between each transmission is chosen to be 400 ms, as also observed in Chapter 5.1.2. Therefore, the total time it takes from the first to the last transmission is 1200 ms. That is the case for both the ledger request and its responses. If all but the last transmission is lost, it will therefore take 2400 ms plus the transmission time from the user sends the request until it receives a response.

By waiting 2500 ms from the request is sent until the user concludes that they are alone in the network, the user can be sure that they will not receive any responses for that ledger request.

5.1.7 Who should send full ledgers?

Every request for a ledger is responded to with either a hash of the ledger or the full ledger. While the full ledgers contain more information and are needed by the new user, they are also more prone to packet loss. The difference in packet loss between hashes and full ledgers increases as the ledger grows. For large ledgers, it is therefore important to restrict how many users send the full ledgers so that as few as possible votes are lost.

Having several users send the full ledger ensures redundancy, although there are several reasons why the full ledger might not be received. The user responsible for sending the full ledger can disconnect from the WLAN, or have closed the application causing them to become unavailable, or the packets sent might be lost. If the full ledger of the accepted ledger is not received, it will have to be requested from one of the users who sent its hash. By having multiple users send the full ledger, this situation is less likely to happen. If the ledger is not completely synchronized, the user who sends the full ledger might also have an outdated version of the ledger, which would also require the correct ledger to be requested.

Based on the results presented in Chapter 5.2.3, we can see that the ledger is rarely unsynchronized, at least for ledgers smaller than 7 LEs. The advantages of

having redundancy in sending full ledgers do not outweigh the increased probability of packet loss in full ledgers, the additional time it takes to send them, and the extra traffic in the network caused by them. Hence, it is reasonable that only one user sends the full ledger while the rest sends hashes.

Which user sends the full ledger and which sends hashes is not of great importance to the ledger's integrity, as both hashes and full ledgers are treated as one vote. However, the responsibility of sending the full ledger must fall to exactly one user. The system cannot know if this user has disconnected, but if that is the case, the ledger will have to be resent by someone else after the voting has taken place.

Sending the full ledger is slightly more energy-consuming than sending just a hash, so avoiding that the same user sends the ledger every time can be advantageous.

It is also an advantage that the user sending the ledger is not malicious and thus is trying to send the correct ledger. It is not crucial to the ledger's integrity, but if the user responsible for sending the ledger does not send it or sends an incorrect one, one of the users that sent the hash of the correct ledger will have to send the full ledger.

The responsibility of sending the full ledger is given to a user with a CA signed certificate, as long as there is such a user in the ledger, to increase the likelihood that the responsible user sends the full ledger correctly. The responsibility is given to the last CA signed user to join the ledger to avoid the same user will have to send the full ledger every time. To know if they should send the full ledger, each user has to keep track of whether someone else has joined the network after themselves. This ensures that everyone knows whether they should send the full ledger or a hash.

5.1.8 Idle time after request

To avoid DoS attack by request flooding, a user waits for some time after responding to a request before the user will respond to new requests. Requests received during this time are discarded. The value for the idle time should be chosen to avoid all transmissions of a request from a legitimate user falling into this window.

Each request is transmitted four times with 400 ms between each transmission, as discussed in Chapters 5.1.1 and 5.1.2. The total time from the first to the last transmission is 1200 ms. At least two transmissions should always fall outside this window to increase the likelihood of receiving legitimate requests. To ensure that that is the case, the idle time must be less than 800 ms. The idle time after responding to a request is therefore set to 790 ms.

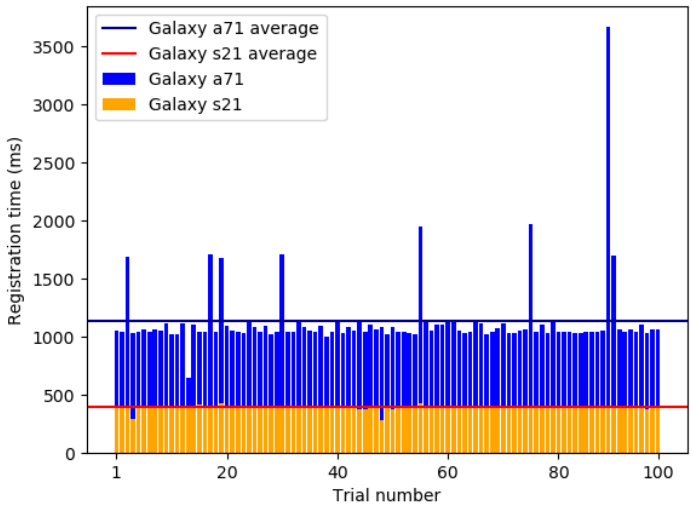


Figure 5.6: Sign up time

5.2 Performance

In the following section, we will assess the performance of the application in terms of the time it takes to sign up, distribute the ledger, and establish a connection, as well as packet loss in UDP packets related to the ledger, and the probability of accepting the correct ledger.

5.2.1 Sign up

For a user to use the application, they will need to sign up. In an offline scenario, the sign-up process includes a check on username availability, generation of a self-signed certificate, and broadcasting the ledger entry. This process should be fast to provide a good user experience. However, the process of checking Internet availability and the generation of certificates are time-consuming tasks and could pose a challenge to the user experience.

A test was performed to find out how long it will take for a user to sign up. The tests start when the user presses the sign-up button in the application and finish when the application has generated an LE for the user. Figure 5.6 shows the sign up times for 100 sign ups using the two phones.

On average, the Samsung Galaxy S21 used 393ms for sign-up, while the Samsung Galaxy A71 used 1136ms. The worst test runs resulted in test times of over 2s, which

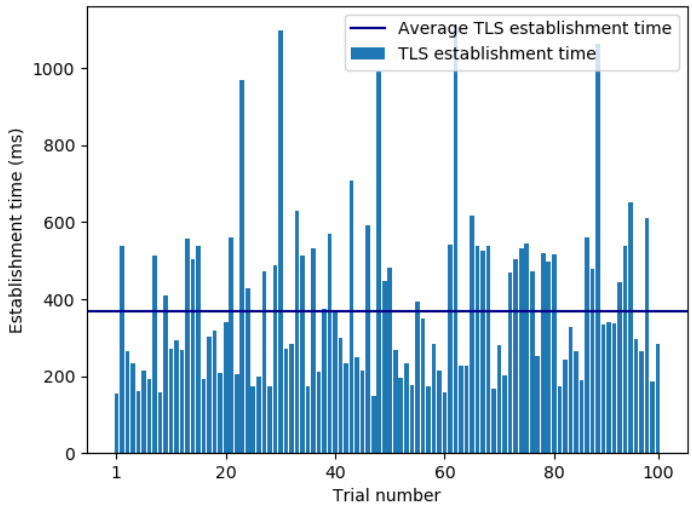


Figure 5.7: mTLS establishment times

is a noticeable amount of time.

In large, the task of signing up depends on the certificate generation time. Generating a certificate is performed within one device and is not dependent on external factors. The Samsung Galaxy S21 is a newer phone running at a clock speed of 2.9GHz [SAM22a], while the Samsung Galaxy A71 runs with a clock speed of 1.8GHz [SAM22b]. That is one of the reasons why the Samsung Galaxy S21 achieves a lower sign-up time compared with the Samsung Galaxy A71.

The second time-consuming task during a sign-up is to check the Internet availability. That is done by pinging Google open DNS at IP-address 8.8.8.8. This task can take up to 500 ms and is contributing to why some of the sign-ups take much time.

5.2.2 Connection establishment time

As described in Chapter 3, the proof of concept application uses a combination of asymmetric and symmetric encryption to achieve a lower connection establishment time. Two tests have been conducted to find the time it takes to establish a connection. One for establishing a mTLS connection, and one for establishing TCP connection. The tests start when the user presses the LE of its peer and finish when a connection has been established. Figure 5.7 shows the connection establishment times for mTLS, and Figure 5.8 shows the connection establishment times for TCP.

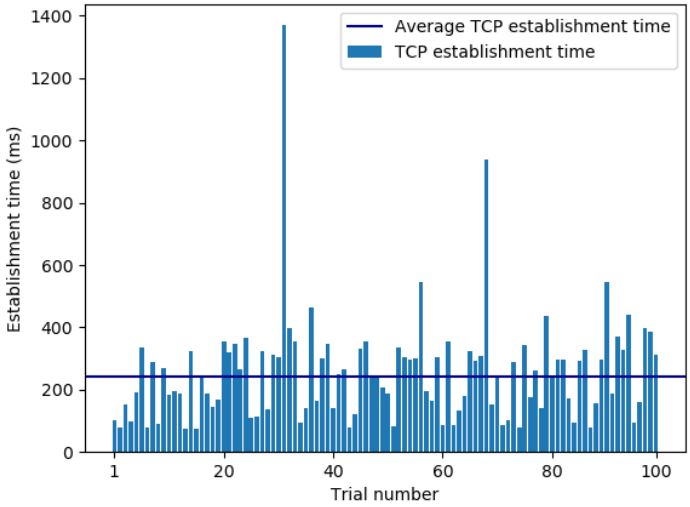


Figure 5.8: TCP establishment times

Test type	Number of trials	Mean	Variance	Confidence interval 95%
Sign up time Galaxy a71	100	1.136s	0.101467	[1.129, 1.142]
Sign up time Galaxy s21	100	0.393s	0.000292	[0.393, 0.393]
mTLS establishment times	100	0.368s	0.045291	[0.363, 0.372]
TCP establishment times	100	0.243s	0.030059	[0.239, 0.246]

Table 5.7: Statistical values for performance tests

The test results show that, on average, it takes 368ms to establish a connection using mTLS and 243ms using TCP.

The test results show that TCP establishes a connection faster than mTLS. However, a difference of 125ms is too small to be notable for a user, and the time saved does not have a significant value to the user. However, with the TCP connection, the first message sent can include a chat message, as no initial handshake is needed. That could be an advantage since messages that carry information valuable to the user are sent earlier. Nevertheless, the user has to type in the message he wants to send. It is reasonable to believe that the typing process is slower than mTLS establishment process of 368ms. Hence, the user will most likely not notice the advantage of being able to send a chat message with the first message sent in the conversation.

5.2.3 Ledger acceptance

Before a user can sign up for the application, the user has to get the correct ledger or conclude that the user is alone in the network. Therefore, the time for accepting the ledger is an important performance metric.

Tests have been conducted to measure how long it takes before the user accepts the ledger. The test starts when the user opens the application and finishes when the application has accepted a ledger. The time it takes to accept a ledger depends on the number of users with CA-signed certificates. When there are one or fewer CA verified users in the network, the ledger acceptance timer described in Chapter 4.2.2 always has to time out before the ledger can be accepted. However, when there are two or more users, they do not need to wait for the timer as long as the conditions described in Chapter 3.3.5 are met.

Figure 5.9 shows the difference in the time it takes to accept the ledger in a network with only CA verified users and a network without any CA verified users. It also shows how the time it takes to accept a ledger drops when the first condition of acceptance is met, and the timer does not have to time out. The drop in acceptance time from one to two CA verified users shows this.

The accepted ledger should contain the LE of all other users in the network. A user has the freedom to communicate with all their peers only if all LEs are included in the ledger. The probability that the n'th user joining the network accepts a ledger containing the LE of all the users that have previously joined can be seen in Table 5.9. The results show that for ledgers up to 5 the accepted ledger are very likely to contain the LE of all the users in the network. However, should the accepted ledger be out-of-date it will be updated next time a new user joins the network.

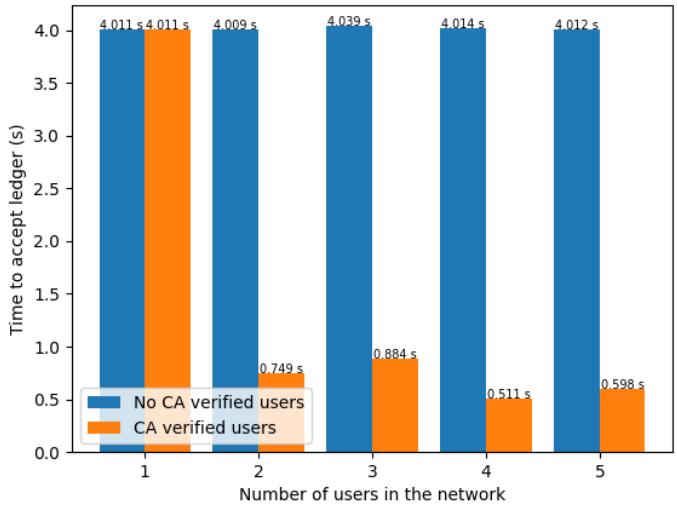


Figure 5.9: The time it takes to accept a ledger is affected by the number of peers in the network

Number of users in network	Number of trials	Mean	Variance	Confidence interval 95%
1	100	4.011s	0.000008	[4.011, 4.011]
2	100	0.749s	0.541153	[0, 1.809]
3	100	0.884s	1.034703	[0, 2.912]
4	100	0.511s	0.420092	[0, 1.335]
5	100	0.598s	0.607002	[0, 1.788]

Table 5.8: Statistical values for time to accept ledger given number of users in network

Number of users in network	Number of trials	Mean	Confidence interval 95%
1	100	1	[0.96378, 1.00000]
2	100	1	[0.96378, 1.00000]
3	100	0.99	[0.94554, 0.99975]
4	100	0.99	[0.94554, 0.99975]
5	100	0.99	[0.94554, 0.99975]

Table 5.9: Statistical values for a correct ledger being accepted with n users in the network

5.2.4 Multicast packet loss

UDP is used as the transport layer protocol for multicast messaging. UDP is a best-effort protocol which means packets may be lost. The loss of multicast packets will affect the system's ability to synchronize the ledger and achieve a consensus on users' identities. Furthermore, as described in Chapter 3.1.2 the system's authentication mechanism relies on consensus on the ledger. Thus, the system's ability to deliver multicast packets is an important metric for performance and security.

To measure multicast packet loss, a test has been designed and conducted. The test measures the probability that a FULL_LEDGER message is lost during a voting. In the test, one user is active in the network as a second user joins. The appearance of the second user triggers a voting where the first user sends a pre-programmed ledger to the multicast group. The size of the pre-programmed ledger is varied, where the values tested are 1, 30, 60, 90, 150 and 180. The packets that are sent and received are recorded.

The tests have been conducted with the optimal variables found in Chapter 5.1. The number of transmissions is 4. The ledger is fragmented with 4 LEs in each fragment. The time between each fragment is 5ms, and between each packet is 400ms.

Figure 5.10 shows the number of FULL_LEDGER messages lost for different sizes of the ledger. Already for a ledger size of 60 LEs, the packet loss is relatively high. However, this is not an upper bound for the number of LEs in the ledger. If the full ledger is not received, the user will request the ledger again and again until it is received successfully, as described in Chapter 3.3.3. The time to learn the ledger increases with the probability of packet loss as users will wait α seconds between each request for the ledger. Hence, the packet loss represents the probability that the ledger must be requested again and that the time to learn the ledger is prolonged.

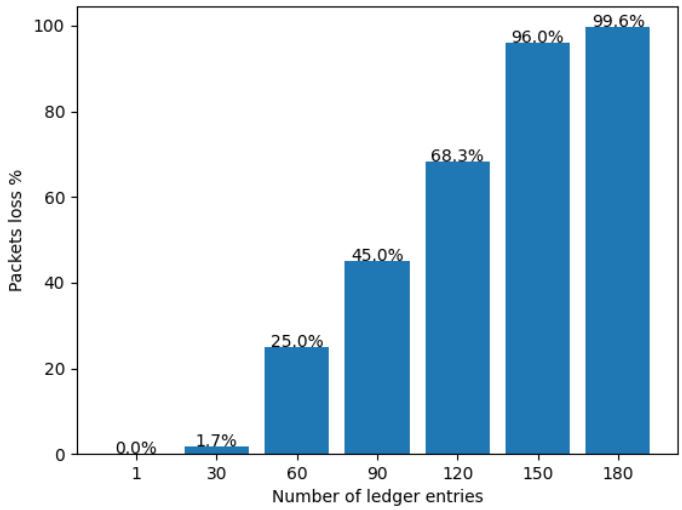


Figure 5.10: Effect on packet loss by size of ledger

Ledger size	Number of trials	Packet loss	Confidence interval 95%
1 LE	300	0.0%	[0.000, 1.222]
30 LEs	300	1.7%	[0.0543, 3.846]
60 LEs	300	25.0%	[20.202, 30.299]
90 LEs	300	45.0%	[39.278, 50.822]
120 LEs	300	68.3%	[62.741, 73.560]
150 LEs	300	96.0%	[93.117, 97.916]
180 LEs	300	99.6%	[98.157, 99.992]

Table 5.10: Statistical values for successful packet delivery given ledger length

Therefore, higher packet loss will increase the number of times the ledger must be requested again. The low packet loss for one LE means that messages sent in one fragment, such as BROADCAST_BLOCK and REQUEST_LEDGER are very likely to be received, which is important to allow users to join the network.

The time it takes to learn the ledger increases when the ledger is sufficiently large, as a user might have to request it multiple times before it is received. This issue can be classified as a scalability issue and be exploited by an attacker to conduct a DoS attack as described in Chapter 6.1.1. This attack would not affect communication among peers already in the ledger but could significantly increase the time it takes for a new user to join.

5.3 Security

To verify the security of the application, packets have been captured with and analyzed with Wireshark [Wir22]. To capture packets, a computer was used as an AP, and all packets were routed through the computer and saved. This section presents the security-related findings.

5.3.1 Message security

When sending unicast messages, the proof of concept application should encrypt the data using TLS or AES depending on the security material previously negotiated. Figure 5.11 shows a Wireshark capture of a data packet the first time two users communicate. As expected, the packet payload is encrypted using TLS and the plaintext can not be read without knowledge of the used encryption key. In TLSv1.3 the version number gets the default value of 0x0303 - "TLS 1.2", as the version number field is not in use [Res18]. Hence, Wireshark shows TLS 1.2 in the version field even when TLSv1.3 is used. The protocol column shows the correct protocol, which is TLSv1.3.

Figure 5.12 shows a Wireshark capture of a data packet the second time two users communicate. As shown in the protocol column, TCP is used as transport layer protocol. TCP does not provide encryption; hence, the data is expected to be in cleartext. However, the payload is encrypted due to the AES encryption on the application layer. The ciphertext is shown in the red box in Figure 5.12. The encrypted payload while using TCP shows that the application works as expected with AES encryption on the application layer the second time two users communicate.

The multicast messages sent from the application are sent in cleartext using UDP. Figure 5.13 shows a multicast message with the BROADCAST_BLOCK message type. The message data is not encrypted and can be read by anyone listening to the

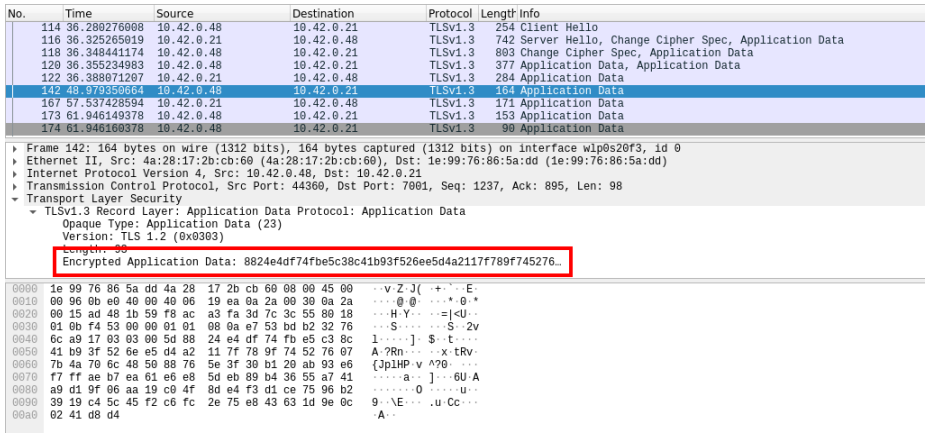


Figure 5.11: Wireshark capture of TLS traffic

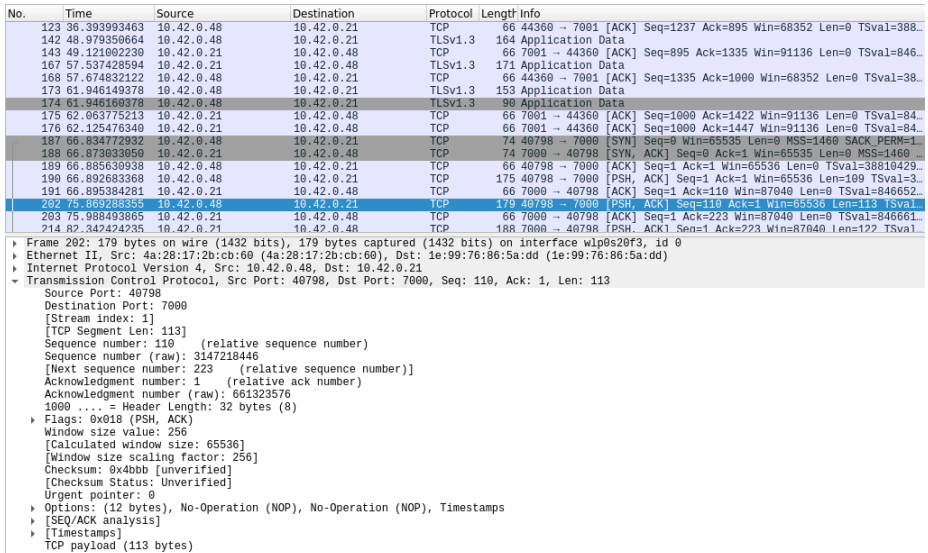


Figure 5.12: Wireshark capture of TCP traffic

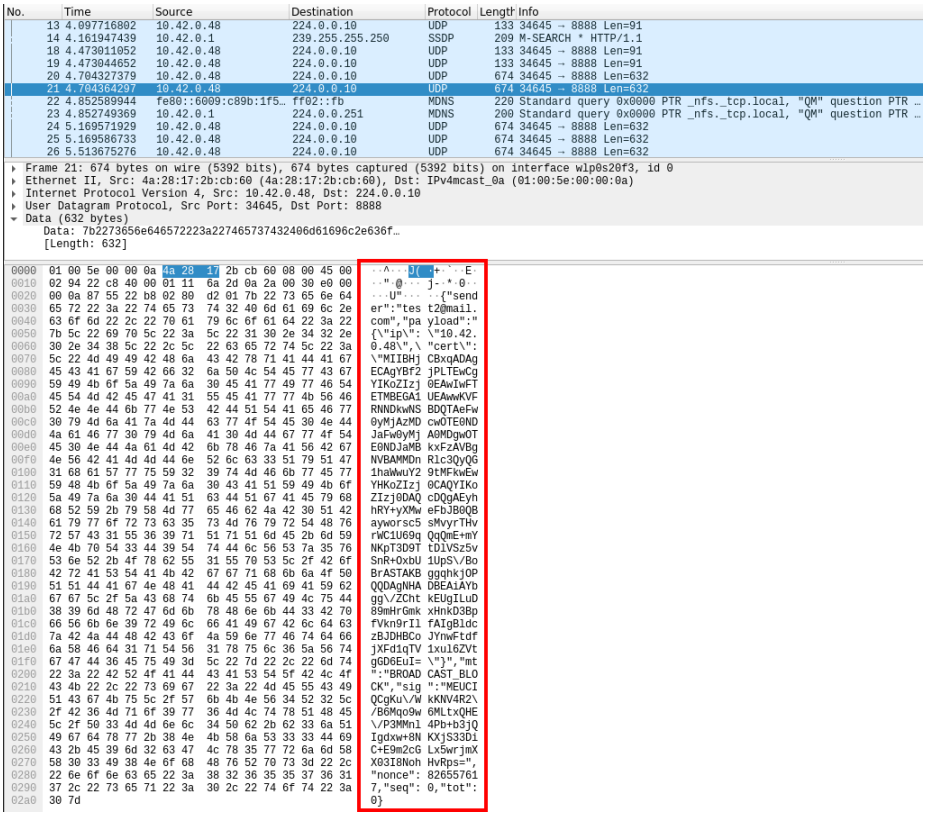


Figure 5.13: Wireshark capture of UDP traffic

multicast group. The message has been signed with the sender's private key, and the signature is appended to the message. The signature ensures message integrity.

Chapter 6
Conclusion

Today a big part of Internet encryption relies on a PKI. In a PKI, the public keys have to be signed by an entity trusted by all parties in order for peers to trust the integrity of the public keys. Unfortunately, networks without Internet access cannot always guarantee the presence of an entity trusted by all parties. Hence, the public keys cannot be trusted, and parties cannot authenticate each other. Furthermore, the absence of a trusted entity makes it challenging to set up a secure communication path without doing out-of-band communication.

This thesis presents a solution for authentication without a central trusted unit using a distributed authentication scheme and public and private cryptographic keys to provide secure communication. The solution is divided into three independent layers. The first layer presents a solution for setting up communication between the peers relying on Wi-Fi infrastructure. The second layer provides authentication by using a distributed ledger. Finally, the third layer provides secure communication relying on mTLS and symmetric encryption.

The main contribution of this work is the distributed authentication scheme. The idea is to create a distributed ledger with authentication material. As the ledger is distributed, all users receive the authentication material needed to authenticate their peers. The consensus mechanism ensures that the network as a whole agrees upon a user's identity and thus the user's authentication material. This solution relocates the responsibility of authentication from one single trusted unit to the network as a cooperation between all network participants.

A proof of concept application has been developed for Android to validate the proposed solution. The application provides secure communication between peers that are connected to the same WLAN.

Various experimental tests have been conducted to identify how different parameters affect the solution's performance. After finding what parameters provide the

best performance, tests of the overall performance of the ledger have been conducted. The tests investigate how the ledger is distributed and how the packet loss changes depending on the ledger size. Additional tests have been performed to investigate the performance of establishing a communication path between devices and activities related to signing up to the system.

The system addresses and solves some authentication issues and establishes a secure communication path in offline networks. However, there are still some drawbacks to the solution. The main drawback is the application's ability to scale when implemented with multicast and UDP as ledger distribution protocol. The application's security increases as more users join the network making it harder for a malicious user to achieve a majority in a voting. However, the system has challenges related to packet loss as the ledger size increases. That means votes may get lost, which may challenge the majority property of the consensus algorithm. In addition, the scaling challenge may open for a DoS attack denying new users to join the network. This attack is further discussed in the further work chapter.

6.1 Further work

The following section introduces suggestions on how the proposed system could be improved. Unfortunately, these suggestions have not been investigated further due to time constraints.

6.1.1 Mitigate ledger exhaustion attack

To join a ledger, a user has to broadcast a valid LE to the network. A malicious actor could quickly generate multiple LEs and add them to the ledger. Because of the scaling issues discussed in Chapter 5.2.4, a sufficiently long ledger could lead to problems for users trying to join the ledger, as high packet loss will increase the time it takes to learn the ledger.

To mitigate this attack, one could, for example, make it harder to join a network or solve the problem with scaling. Making it harder to join the network could be made by requiring a Proof-of-Work. However, the additional work may increase the sign-up time and affect the user experience. Work that would significantly affect the time it takes to sign up on mobile devices would not sufficiently slow down computers, meaning that an attack could still be conducted.

Another way to mitigate the attack is to check that a human conducts the sign-up process. That could be done automatically or by proving the sign-up to another user, which would have to vouch for the new user.

However, the preferred way to mitigate this attack is to solve the problem with large ledgers. One solution that could potentially solve the problem is proposed in Chapter 6.1.3.

6.1.2 Recording user activity

The proposed solution makes no effort to record or evaluate user activity or behavior. The only indication of a user's trustworthiness is their certificate. That means a user who acts maliciously or attacks the network is treated the same way as a regular user. Many of the packets sent in the system are signed, allowing the system to know who created the packet. Recording and evaluating a user's activity could determine their intention and, if needed, penalize them.

Such a solution could be implemented in an offline and online environment. By sharing this information in an offline environment, other users could use this information to determine whether to communicate with this user or not. However, this information would only be available to the users in the current network. If the malicious user is kicked out of the network, they could move to a new network and continue the malicious activity. The new network would not know of the malicious activity performed in the previous network. Thus, this information should be uploaded to a central trusted unit when the system goes online.

The CA could use the uploaded user activity to determine what penalties to put on the malicious user. For example, the malicious activity could be punished by not giving out a certificate or adding a score to the certificate indicating the user's trustworthiness. The trustworthiness score could be used to reduce the user's influence during voting or to indicate to other users how to relate to this user. In the current system, the certificate only needs to be renewed once a year, giving a malicious actor a long time to do the malicious activity before the consequences are reflected in the user's certificate. That can be handled by reducing the certificate validity time.

6.1.3 Increased use of TCP

Broadcasting the messages regarding the ledger has its benefits, as everyone can listen to all the traffic. Hence, the users do not have to rely on others to forward information to them. It does, however, force the service to use UDP as transport layer protocol.

TCP for full ledgers

UDP has brought us several advantages like multicast and being lightweight, but it also comes with some drawbacks regarding an unreliable packet delivery. As

discovered, the packet loss is also exacerbated by an increased ledger size. A high packet loss increases the chances that a vote is lost and that a packet must be retransmitted, affecting the time it takes to learn the ledger and sign up.

When a ledger is lost, it is likely that only some of the fragments are lost. Thus, requiring that the full ledger is sent again can be inefficient. TCP offers mechanisms for only requesting the lost fragments, e.g., through selective acknowledgments, which could be utilized when sending full ledgers. Hence, the solution could avoid sending the full ledger over UDP to increase efficiency and performance when the ledger increases in size. Instead, every user could send the hash of the ledger over UDP, and the users that do not hold the correct ledger could request the full ledger over TCP.

Partial mesh topology using TCP

To send messages to all the other users using TCP would require every user to maintain a TCP connection to every other user. That would be resource-intensive and not scale well with many users.

In a partial mesh topology, each user is connected to a number of its peers. The solution could replace broadcasting messages using UDP with creating a TCP based partial mesh network, where the users are responsible for relaying information it receives to their peers. Because this solution would be based on TCP, the transport of packets would be lossless. However, each user would have to trust that at least one of its peers relays the correct packets. Therefore, this solution would have to include measures for setting up the network and maintaining the connections between peers and consider how many peers each user should be connected to directly.

References

[ADF+10] K. Akdemir, M. Dixon, *et al.*, «Breakthrough aes performance with intel aes new instructions», Intel, Tech. Rep. DOE-SLC-6903-1, 2010.

[Adv16] U. G. C. S. Adviser. «Distributed ledger technology: Beyond block chain». (2016), [Online]. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf (last visited: Mar. 15, 2022).

[AK21a] S. Aggarwal and N. Kumar, «Chapter eleven - cryptographic consensus mechanisms introduction to blockchain.», in *The Blockchain Technology for Secure and Smart Applications across Industry Verticals*, ser. Advances in Computers, S. Aggarwal, N. Kumar, and P. Raj, Eds., vol. 121, Elsevier, 2021, pp. 211–226. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245820300668.

[AK21b] ——, «Chapter twenty - attacks on blockchain», in *The Blockchain Technology for Secure and Smart Applications across Industry Verticals*, ser. Advances in Computers, S. Aggarwal, N. Kumar, and P. Raj, Eds., vol. 121, Elsevier, 2021, pp. 399–410. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245820300759.

[akw22] akwizgran, *A quick overview of the protocol stack*, 2022. [Online]. Available: https://code.briarproject.org/briar/briar/-/wikis/A-Quick-Overview-of-the-Protocol-Stack.

[All16] W.-F. Alliance, *Wi-fi peer-to-peer (p2p) technical specification version 1.7*, 2016. [Online]. Available: https://documents.pub/download/wi-fi-p2p-technical-specification-technical-specification-version-17-this-document.

[All21] ——, *Wi-fi aware*, 2021. [Online]. Available: https://www.wi-fi.org/discover-wi-fi/wi-fi-aware.

[And22] Android. «Android keystore system». (2022), [Online]. Available: https://developer.android.com/training/articles/keystore (last visited: May 12, 2022).

[BGM21] M. Bansal, S. Gupta, and S. Mathur, «Comparison of ecc and rsa algorithm with dna encoding for iot security», in *2021 6th International Conference on Inventive Computation Technologies (ICICT)*, 2021, pp. 1340–1343.

[BMTJ21] S. Berbom, E. M. Mæland, *et al.*, *Keeping connected without the internet backbone*, Dec. 2021.

[Bri22] Bridgify, *Faq bridgify*, 2022. [Online]. Available: https://bridgefy.me/faq/.

[But14] V. Buterin. «Ethereum: A next-generation smart contract and decentralized application platform». (2014), [Online]. Available: https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf (last visited: Jun. 7, 2022).

[CGS13] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, «Device-to-device communications with wi-fi direct: Overview and experimentation», *IEEE Wireless Communications*, vol. 20, no. 3, pp. 96–104, 2013.

[CSF+08] D. Cooper, S. Santesson, *et al.*, «Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile», IETF, RFC 5280, May 2008. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5280.

[CY12] F. Chen and J. Yuan, «Enhanced key derivation function of hmac-sha-256 algorithm in lte network», in *2012 Fourth International Conference on Multimedia Information Networking and Security*, 2012, pp. 15–18.

[Dan12] Q. Dang, *Recommendation for applications using approved hash algorithms*, en, Aug. 2012. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=911479.

[Gar07] V. K. Garg, «Chapter 21 - wireless local area networks», in *Wireless Communications & Networking*, ser. The Morgan Kaufmann Series in Networking, V. K. Garg, Ed., Burlington: Morgan Kaufmann, 2007, pp. 713–776. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123735805500557.

[Goo21a] Google. «Android 11». (2021), [Online]. Available: https://www.android.com/android-11/ (last visited: Oct. 29, 2021).

[Goo21b] ——, «Sdk platform release notes». (2021), [Online]. Available: https://developer.android.com/studio/releases/platforms (last visited: Oct. 29, 2021).

[Goo22a] ——, «Cryptography». (2022), [Online]. Available: https://developer.android.com/guide/topics/security/cryptography (last visited: May 12, 2022).

[Goo22b] ——, «Firebase documentation». (2022), [Online]. Available: https://firebase.google.com/docs?gclsrc=ds&gclsrc=ds&gclid=CJiBheC32fcCFQ3TGQodmS0Ktg (last visited: May 12, 2022).

[HHBS18] M. T. Hammi, B. Hammi, *et al.*, «Bubbles of trust: A decentralized blockchain-based authentication system for iot», *Computers & Security*, vol. 78, pp. 126–142, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818300890.

[HN17] J. Huang and D. Nicol, «An anatomy of trust in public key infrastructure», *International Journal of Critical Infrastructures*, vol. 13, p. 238, Jan. 2017.

[IEE21] IEEE, «Ieee standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications», *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pp. 1–4379, 2021.

[IET17] I. E. T. F. (IETF). «Udp usage guidelines». (2017), [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8085 (last visited: May 31, 2022).

[Jet21] JetBrains. «Kotlin». (2021), [Online]. Available: https://kotlinlang.org/ (last visited: Oct. 29, 2021).

[JKSS12] T. Jager, F. Kohlar, *et al.*, «On the security of tls-dhe in the standard model», in *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 273–293.

[KBC97] H. Krawczyk, M. Bekkare, and R. Canetti, «Hmac: Keyed-hashing for message authentication», IETF, RFC 2104, Feb. 1997. [Online]. Available: https://www.ietf.org/rfc/rfc2104.txt.

[KR13] J. Kurose and K. Ross, «Computer networking : A top-down approach», in *Computer networking : a top-down approach*. Pearson, 2013, pp. 676–699.

[KW05] J. Korhonen and Y. Wang, «Effect of packet size on loss rate and delay in wireless links», in *IEEE Wireless Communications and Networking Conference, 2005*, vol. 3, 2005, 1608–1613 Vol. 3.

[LAM05] M. Leon, R. Aldeco, and S. Merino, «Performance analysis of the confidentiality security service in the ieee 802.11 using wep, aes-ccm, and ecc», in *2005 2nd International Conference on Electrical and Electronics Engineering*, 2005, pp. 52–55.

[LFF20] X. Liu, B. Farahani, and F. Firouzi, «Distributed ledger technology», in *Intelligent Internet of Things: From Device to Fog and Cloud*. Cham: Springer International Publishing, 2020, pp. 393–431. [Online]. Available: https://doi.org/10.1007/978-3-030-30367-9_8.

[MP16a] M. Marlinspike and T. Perrin, «The double ratchet algorithm», Open Whisper System, Tech. Rep., Nov. 2016.

[MP16b] ——, «The x3dh key agreement protocol», Open Whisper System, Tech. Rep., Nov. 2016.

[MW00] U. Maurer and S. Wolf, «The diffie–hellman protocol», in *Designs, Codes and Cryptography 19*, 2000, pp. 147–171.

[Nak09] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf.

[Onl22] O. Online. «Oxford university press». (2022), [Online]. Available: https://www.oed.com/view/Entry/13320?rskey=itQ9tf&result=2#eid (last visited: Apr. 4, 2022).

[PMC20] S. Popov, H. Moog, and D. Camargo, *The coordicide*, 2020. [Online]. Available: https://files.iota.org/papers/20200120_Coordicide_WP.pdf.

[Res18] E. Rescorla, «The transport layer security (tls) protocol version 1.3», IETF, RFC 8446, Aug. 2018. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8446.

[RY06] R. A. Rashid and R. Yusoff, «Bluetooth performance analysis in personal area network (pan)», in *2006 International RF and Microwave Conference*, 2006, pp. 393–397.

[SAM22a] L. SAMSUNG ELECTRONICS CO. «Specifications galaxy s21 fe | s21 | s21+ 5g». (2022), [Online]. Available: https://www.samsung.com/global/galaxy/galaxy-s21-5g/specs/ (last visited: May 31, 2022).

[SAM22b] ——, «Specifications galaxy s21 fe | s21 | s21+ 5g». (2022), [Online]. Available: https://www.samsung.com/no/business/smartphones/galaxy-a/galaxy-a71-a715-sm-a715fzkunee/ (last visited: May 31, 2022).

[SSTJ20] K. L. Skaug, E. B. Smebye, *et al.*, «Keeping connected in internet-isolated location», M.S. thesis, Norwegian University of Science and Technology, 2020.

[Sta16] W. Stallings, «Cryptography and network security: Principles and practice, global edition», in *Cryptography and Network Security: Principles and Practice, Global Edition*. Pearson Education Limited, 2016.

[Sta22] Statista, *Whatsapp - statistics & facts*, 2022. [Online]. Available: https://www.statista.com/topics/2018/whatsapp/.

[STJ19] Ø. L. Sigholt, B. Tola, and Y. Jiang, «Keeping connected when the mobile social netowrk goes offline», M.S. thesis, Norwegian University of Science and Technology, 2019.

[Sun20] A. Sunyaev, «Distributed ledger technology», in *Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies*. Cham: Springer International Publishing, 2020, pp. 265–299. [Online]. Available: https://doi.org/10.1007/978-3-030-34957-8_9.

[The21] The Legion of the Bouncy Castle. «Bouncycastle». (2021), [Online]. Available: https://www.bouncycastle.org/ (last visited: Oct. 29, 2021).

[Wha21] WhatsApp, «Whatsapp encryption overview», WhatsApp, Tech. Rep., Sep. 2021.

[Wha22] ——, *Whatsapp security*, 2022. [Online]. Available: https://www.whatsapp.com/security.

[Wir22] Wireshark. «Wireshark». (2022), [Online]. Available: https://www.wireshark.org/ (last visited: May 12, 2022).

[Woo20] M. Woolley, «Bluetooth mesh networking - an introduction for developers», Bluetooth, Tech. Rep., Dec. 2020.

Appendix A
Scientific Paper

A scientific paper which covers the proposed core concept has been produced. It will soon be submitted to the IEEE Conference on Communications and Network Security the 26–28 September 2022, and has been included on the following pages.

Distributed Trust Empowerment for Secure Offline Communications

Endre Medhus Mæland, Sigmund Bernhard Berbom, Besmir Tola, Yuming Jiang
Department of Information Security and Communication Technology
NTNU, Norwegian University of Science and Technology

***Abstract*— Most of today's digital communications over the Internet rely on central entities, such as certificate authority servers, to provide secure and authenticated communication. In situations when the Internet is unavailable due to lack of reception in remote areas, natural disasters destroying network infrastructure, or congestion due to large amounts of traffic, these central entities may not be available. This causes secure communication, even among users in the vicinity of each other, to become a challenge. This paper contributes with a solution that enables peers within the vicinity to communicate securely without a connection to the Internet backbone. The solution operates on the Wi-Fi infrastructure mode and exploits a private distributed ledger to ensure a trusted authorization among users without a third party. Moreover, the solution enables users to set up secure communication channels using mutual authentication for exchanging data securely. Finally, the solution is validated through a proof of concept application and extensive experiments aiming at optimizing system parameters and investigating the performance of the application are carried out. The results from these measurements indicate that the solution performs well on small to medium-scale networks.**

***Index Terms*—Decentralized Authentication, Distributed Ledger Technology, Device-to-Device Communication, Peer-to-Peer Network, mTLS, Mobile Social Network.**

I. Introduction

The use of certificates and private/public key pairs is a common way to provide authentication over the Internet [1]. Certificates enable end users to be authenticated from a centralized Certification Authority (CA), a trusted third party, in order to retrieve service [2]. However, the process of authentication is challenged in situations where there is a lack of Internet access or backbone connectivity. In case natural disasters, power outages, or human-caused accidents impact the Internet infrastructure, end users will not be able to retrieve authentication and consequently use a secure service, or establish secure communication.

Although the Internet may be unavailable due to disaster impacts, mobile-equipped end users can still establish network connectivity within the range of their mobile radios. Almost all modern mobile devices are equipped with Wi-Fi radios and this technology can be exploited for re-establishing connectivity in a Peer-to-peer (P2P) fashion for offline communication. However, the lack of a trusted third party will still prevent the users from establishing and providing secure and authenticated communication. Wi-Fi can provide authentication through WPA-Enterprise [3]. However, WPA-Enterprise requires an authentication server, such as a RADIUS server, for establishing user identities but if the server is not available due to Internet connectivity issues, Wi-Fi cannot provide authentication. Furthermore, in a scenario without access to the Internet, the users in the vicinity will not be able to establish a secure communication channel due to the lack of a central unit orchestrating the communication. As a result, additional security mechanisms need to be built on upper layers to enable data confidentiality, integrity and authenticity.

A distributed authentication mechanism can solve the issue of a CA not being available on the network by allowing benign nodes to agree on an immutable record of authentication material, despite the existence of malicious nodes. To achieve such an agreement, Distributed Ledger Technologies (DLTs) can be utilized [4]. However, most public DLTs have resource-consuming security mechanisms tailored for financial transactions. Transaction are typically secured by consensus mechanisms such as Proof-of-Work (POW) and Proof-of-Stake (POS), requiring large amount of resources or financial transactions [5]. Therefore, these consensus mechanisms are unsuitable for resource-scarce mobile devices when exchanging authentication material. The mutual authentication in mutual Transport Layer Security (mTLS) [6] allows all parties to be authenticated, and the protocol also provides data encryption. Furthermore, using symmetric encryption combined with mTLS can improve connection establishment times compared to only using mTLS.

A previous work addressing authentication in offline networks proposes a system where users receive authentication material when an Internet connection is available [7]. If the user loses the Internet connection, the authentication material could still be used. However, this solution does not support offline registrations, hence preventing new users to be authenticated after the loss of Internet connection. [8] improves this solution by using peer signed certificates to allow offline registration. The peer signed certificates establish a trust chain where users vouch for each other. In this solution, it is difficult to draw a clear line to what point in the chain a user is no longer trusted. In [9], the authors propose a blockchain-based solution for decentralized authentication of IoT devices. The proposal is based on the public blockchain Ethereum and envisions the creation of specific trusted zones, called Bubbles, where IoT devices within the zone establish a level of trust. The trust is confined within each zone and inter-zone authentication and trust is left for future work. However, as also identified by the authors, the solution presents several

open issues related to the use of a public DLT. The solution is associated with economic cost, is unsuited for real-time applications, and requires an initialization phase with a node assuming the role of a certification authority. Such solution is infeasible for use cases with high user mobility. Henceforth, a problem yet to be solved is *how to efficiently authenticate users in a mobile offline environment where users are able to register offline, establish a secure communication path,* and have a clear separation between trusted and untrusted users.

This paper aims to design, implement, and validate a solution for application layer security in offline networks. The proposed solution exploits Wi-Fi Infrastructure mode as the technology for offline communication, mTLS in combination with a symmetric key solution to provide mutual authentication and encrypted data exchange among peers, and implements a private distributed ledger and a consensus mechanism to agree on users' authentication material. A proof-of-concept instant messaging application has been developed to validate the solution and the implementation source code is publicly available[1]. An extensive experimental campaign on real equipment has been performed for optimizing the system parameters and analyzing its performance and security features.

The remainder of the paper is structured as follows: Section II illustrates the proposed system architecture for enabling secure and trustworthy communication over Wi-Fi Infrastructure mode. The implementation on a real testbed of smart devices running the Android OS is presented in Section III. Successively, the validation of the security adopted in the architecture and the analysis of the experimental results are presented in Section IV. Finally, Section V concludes the paper.

II. Proposed solution

This section illustrates how the proposed solution enables authenticated and secure communication.

A. Overall Architecture

Figure 1 illustrates the high-level architecture of the proposed solution.

Going bottom-up, the first layer is the wireless connection layer. This layer exploits Wi-Fi in infrastructure mode and is responsible for handling wireless connections, including both unicast and multicast transmissions utilized for ledger management and the actual service. This layer can also be built on other technologies but we chose Wi-Fi given the wide adoption of Wi-Fi radios in mobile devices.

The second layer is responsible for handling identities and enabling authentication. This layer consists of the ledger and the consensus mechanism. The ledger contains authentication material for all users in the network. By having all users agree on the ledger's content, the responsibility of authenticating users is moved from a single entity to the network as a whole. This layer is further discussed in the following sections.

The third layer is responsible for enabling secure communication. The first time two users connect, an mTLS connection will be established with the authentication material found in the ledger. During this connection, the users will negotiate a symmetric key using the Diffie-Hellman key exchange [10] to be used the next time they communicate. From the next time these users connect, they will establish a TCP connection secured with AES in Galois/Counter Mode (GCM) [11]. Using symmetric keys reduces the time to establish the connection. To improve the security of the symmetric keys, DH- and symmetric ratchets are used [12].

[1]Will be disclosed at a later time.

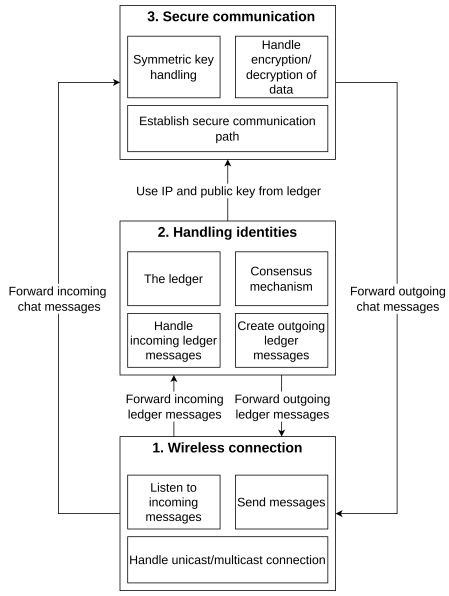


Fig. 1: High-level architecture of the solution.

B. Distributed Authentication

This subsection describes how a ledger is used to achieve distributed authentication.

1) Ledger Entry: The ledger consists of Ledger Entries (LEs), where one LE represents one user in the network. When new users want to join a network, they have to create a valid LE and distribute it. Each LE contains an X.509 certificate [13] and an IP address. The certificate can be signed by a CA or by the users themselves. To obtain a CA-signed certificate, the user has to sign up to the system while the application is online using a password and an email address as the username. The email address must be verified before receiving the CA-signed certificate. Whether the certificate is CA- or self-signed plays an important role in the consensus mechanism. This is further described in Section II-B3.

2) Joining the ledger: When a new device connects to a network, it first joins a predefined multicast group to which all users attached to the network are listening. Secondly, it broadcasts a request for the ledger to all the users in this group. If the user does not receive any responses within α seconds, the user assumes there are no other users in the network. The

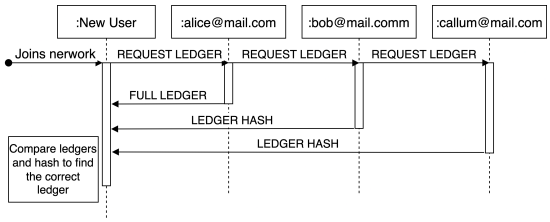


Fig. 2: Multicast message exchange when a new user joins the network.

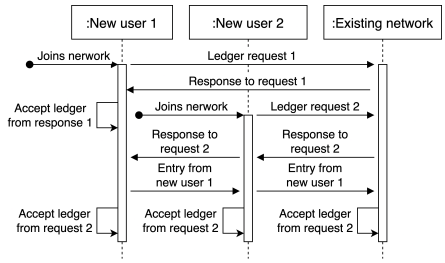


Fig. 3: LE of new user 1 is not included in the ledger from request 2, and is therefore lost after new user 1 has registered.

user will then proceed to create its own LE, which at this point will make up the ledger. The value of α is further discussed in Section IV-A6.

Figure 2 illustrates the process of a new user joining an already established network. The last CA-signed user to join the ledger will respond by sending a full ledger, while the others will send the hash of their ledger. If there are no CA-signed users in the network, the last user to join the network will send the full ledger. The new user will use these responses to select the correct ledger according to the criteria described in Section II-B3.

All messages regarding the ledger are sent using UDP multicast. If the user has not received the full ledger that won the voting, the user will randomly pick one of the users who sent the hash of the winning ledger and request the full ledger. If the user does not receive the full ledger within β seconds, this process is repeated until the full ledger is received. The value of β is further discussed in Section IV-A5.

After receiving the ledger, the user can create and broadcast its own LE. The username used by the new user cannot already exist in the ledger unless the new user can obtain a CA-signed certificate with that username. If so, the LE with the CA-signed certificate will replace the one with the self-signed certificate, and the old user will no longer exist in the ledger.

3) Consensus in the ledger: When a new user joins the network, a voting is initiated to agree upon and distribute the ledger to the new user. As shown in Figure 2, all users send a full ledger or a hash of their ledger. These messages are interpreted as votes for the validation of the correct ledger by the new user. The messages have to be signed, and their certificate must be included to ensure that each user can vote only once. A malicious user could generate many fake users with self-signed certificates, corrupt the ledger with fake entries, and drive the consensus. This would enable a Sybil attack [14] whose consequences would be a Denial of Service (DoS). To mitigate this, votes from users with CA-signed certificates are given priority. This is because CA-signed certificates contain usernames, i.e., email addresses, that have to be validated online thus making the process of impersonation much harder. As a result, the following restrictive criteria, with a decreasing priority, must be met before a user accepts a ledger:

1) If at least two CA-certified users distribute the same ledger or corresponding hash, and they make up more than 50% of the CA-certified users in the ledger, that ledger will be accepted, given the full ledger has been received.
2) If β seconds have passed from the time the ledger was requested and at least one CA-certified user has responded with the ledger, either full or hashed, the ledger with most votes from CA-certified users is accepted.
3) If β seconds have passed from the ledger was requested and no CA-certified users have responded with the ledger, the ledger with most votes will be accepted.

β has to be set so that users can expect to have received all the votes within that time. This value is discussed in Section IV-A. Note that the above criteria are not exclusive of each other. Priority is given to CA-signed users, however, self-signed users are considered in some scenarios to ensure the service is available even though no CA-signed users are present. This may pose a risk in scenarios without CA-signed users, but the risk will decrease as the number of user increases.

4) Synchronizing the ledger: When a new user joins the network, all users can listen to the following voting process because all messages are broadcast. They can therefore see which ledger is correct by comparing responses, in the same way as the user joining. Hence, users with an incorrect ledger can update their ledger. If the accepted ledger has LEs that does not exist in the user's ledger, they are added. If a user's ledger holds any LEs that do not exist in the accepted ledger, those LEs are not removed. This mechanism combats a possible ledger rollback attack, further mentioned in Section II-B5, and ensures no LEs are lost. Figure 3 shows how a LE could potentially have been lost if two users joined at the same time.

An LE conflict means that two LEs with different certificates have the same username. If there are conflicting LEs between the accepted ledger and the ledger held by a user, the users will update their ledger as long as the conflicting LE they hold does not have a CA-signed certificate.

5) Mitigating attacks: There are several security mechanisms added to the proposed solution to mitigate various types of security attacks.

If a valid vote in one voting process could also be valid in another voting, a malicious actor could exploit this to execute

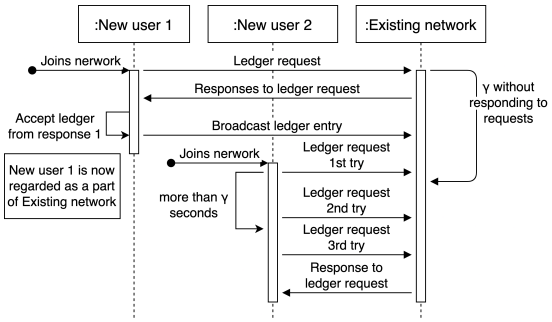


Fig. 4: If one ledger request falls within the time window where a request is dropped, another will fall outside it.

a replay attack. To mitigate this, every voting has a unique nonce where all messages related to this vote have to contain this nonce. Because the messages are signed, and an attacker cannot forge a signature, there is no way for them to obtain a valid response signed by another user, and the attack is prevented from occurring.

An extension to the above attack is a ledger rollback where an attacker stores both the request for the ledger and its responses. As a result, the ledger could be reset to a previous state by replaying these messages later, potentially removing users from the ledger. To avoid this from happening, LEs are not removed from the ledger even when they are not a part of the accepted ledger.

Every request for the ledger broadcast in the network triggers a response from the other users. All the users in the network then handle these responses. Therefore, flooding the network with requests will cause an increase in computational load on the devices, potentially leading to a DoS attack. To combat this attack, a user will drop requests received within γ seconds after responding to a request for the ledger. The choice of the parameter γ is discussed in Section IV-A7. Multicast packets are transmitted multiple times to avoid packet loss. As long as γ is less than the time between the first and last request transmission, dropping these packets will not lead to requests not getting a response, as also shown in Figure 4.

C. Secure communication path

The first time two users connect, they establish an mTLS connection using the authentication material found in the ledger. During this connection, the peers negotiate a symmetric key to be used for the next connection. The second time two peers communicate, they use a pure TCP connection with AES for encryption. Every time they communicate, the peers negotiate a key they will use the next time they communicate.

Diffie-Hellman (DH) key exchange is used to negotiate the symmetric keys. By introducing DH-ratchet, this process gets faster as two messages are needed the first time two users communicate, while only one message is required from there on. The process is illustrated in Figure 5. The first DH key exchange between two users initiates a DH-ratchet where one

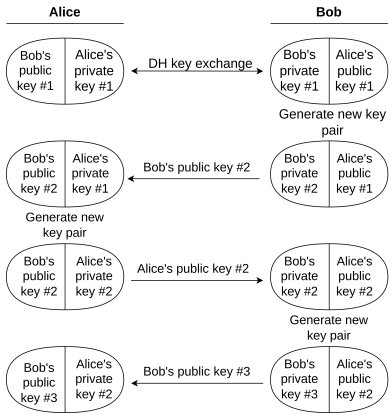


Fig. 5: DH-ratchet

user's private key and the other user's public key are used to calculate a shared DH secret. From the second time the two users interact, only one user updates their key pair to generate the new symmetric key. Therefore, only one key is sent, and only one message is required to update the symmetric key. The users renegotiate the symmetric key every time they set up a new connection.

If an attacker can break one of the symmetric keys, they will be able to read all messages within that conversation. By introducing double ratchets, this problem is reduced to a backward secrecy problem. Double ratchets use a key derivation function (KDF) on every key after use to ensure a key is only used once. Hence, it is not possible to find an old key given a new one, but it is possible to find a new key given an old one. Using double ratchets and DH-ratchets provides forward secrecy within a conversation and forward- and backward secrecy between conversations [12].

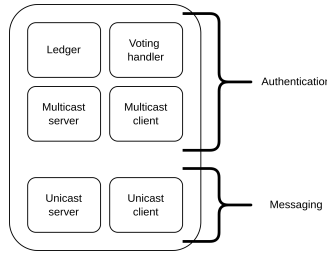


Fig. 6: Application overview.

III. Proof-of-Concept Implementation

In order to validate the proposed solution, the system has been implemented in a proof-of-concept Instant Messaging application. The application consists of the six components, shown in Figure 6.

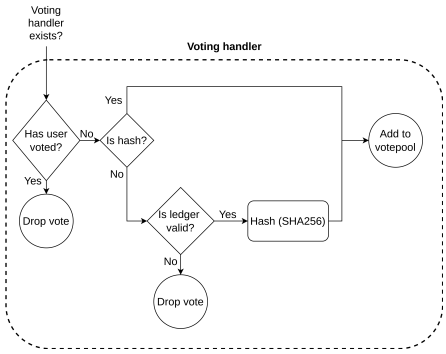


Fig. 7: Voting handler flowchart.

A. Authentication components

The multicast client and server are responsible for sending messages related to the ledger. The multicast server is implemented as an Android Service while the client is a Kotlin class. Both are initiated when the application starts.

The voting handler is responsible for handling the process related to achieving consensus on the ledger. For every new voting, the application initiates a new voting handler. The voting handler ensures the votes are in the correct format, counts votes, selects the correct ledger, and is responsible for updating the ledger after a finished voting.

When a new vote is received by the multicast server, it checks if there exists a voting handler for that nonce, and if so, forwards it to the correct handler. Figure 7 shows how a new vote is processed by the voting handler. The process ensures the vote is related to an existing voting, that a user does not vote multiple times, and ensures the vote is in the correct format.

The ledger module contains the ledger itself and methods for updating it and creating new LEs. The application updates the information in the ledger after every voting.

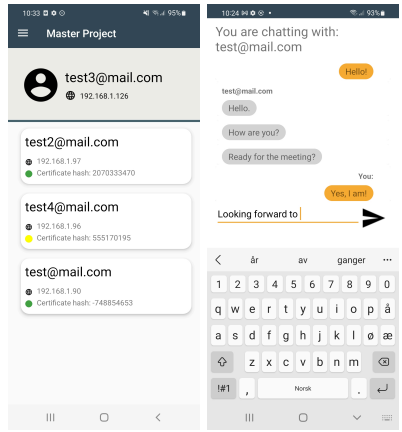
B. Messaging components

When the application starts, it initiates a unicast server. The unicast server listens for incoming requests to set up unicast communication. The server opens a new port for every new connection request, allowing the application to support receiving messages from multiple devices simultaneously. Depending on the available authentication material, an mTLS over TCP or pure TCP connection is established.

When the user starts a chat, the application initiates a unicast client. The unicast client sends a connection request to its peer's unicast server and establishes a connection. After a unicast connection is established, both server and client send and receive messages.

C. Activities

Android divides applications into different interfaces called activities. The proof-of-concept application consists of two primary activities: the main and the chat activity.



(a) Main Activity. (b) Chat Activity.

Fig. 8: Android Application Activities.

Figure 8(a) shows the main activity interface. The interface displays the content of the ledger as a list of users. Each list entry displays the username, IP address, and a colored dot indicating if the user's certificate is self-signed (yellow) or CA-signed (green). The application will initiate a chat with a peer when the user taps a list entry. The application does not limit the users' ability to connect to other users based on their certificate type. The users themselves have to decide whether or not to trust a user with a self-signed certificate.

When a connection between two peers has been initiated, the chat activity shown in Figure 8(b) starts. In this activity, the users can read and send messages.

D. Ledger design parameters

The system must achieve a consensus on the ledger content to enable authentication. The ledger distribution can be challenged by the use of unreliable access and transport layers, i.e., wireless channel and UDP. Henceforth, the following parameters are used to tune mechanisms on the application layer that have been added to address the eventual packet loss: i) *number of transmissions*; ii) *time between transmissions*; iii) *fragment size*; and iv) *time between fragments*. A description of each of the parameters is provided below, while tests used for optimizing the value of each parameter can be found in Section IV.

i) *Number of transmissions*: All messages used for ledger management are sent multiple times to reduce packet loss through redundancy.

ii) *Time between transmissions*: The time between transmissions is defined as the time from sending the last fragment of the ledger in one transmission until sending the first fragment of the ledger in the subsequent transmission. Increasing this parameter may reduce the probability that the same factor, e.g., propagation conditions, will affect multiple transmissions. However, too much time between packets will increase the

time it takes to sign up, potentially affecting the user experience.

iii) *Fragment size*: When a sufficiently large ledger is sent, the message is divided into several fragments. In the proof-of-concept application, one fragment holds n LEs. Every fragment must be received for the ledger to be counted as a vote. When the size of each fragment is reduced, the packet loss is also reduced as found in [15]. However, with smaller fragments the number of fragments that has to be successfully received increases.

iv) *Time between fragments* When a ledger is fragmented, the time between fragments may affect the packet loss similarly to the time between transmissions.

IV. Results

This section describes the test conducted to evaluate the ledger parameter values and assess the security and performance of the proof of concept application.

A. Optimizing parameters

Specific tests have been conducted for optimizing the parameters identified in Section III. In the tests, one user sends a pre-programmed ledger to another user, and the messages sent and received are recorded. The tests only regard message loss, so there is no need for multiple peers to vote for a correct ledger. Each ledger is seen as one message, although it is transmitted in fragments to reduce message loss, with each fragment containing 1 LE. Therefore, a ledger is considered lost if one or more fragments are not received. That is because the entire ledger will have to be received in order for it to be considered a vote. The tests have been conducted using two Samsung Galaxy S21 5G phones and a Netgear Nighthawk M2 wireless router. The optimal value of each parameter has been found by varying its value, while keeping the parameters fixed. The fixed parameters' values are chosen to be near-optimal, based on a smaller sample of the performed tests.

The test environment of the following tests is constructed to have a large message loss to increase the statistical significance. As the number of ledger entries in the pre-programmed ledger negatively affects the message loss, the number of LEs is chosen so that the message loss will be near 50% for what is believed to be the optimal value for the tested parameter. Such message loss will increase the statistical significance of the results compared to a very high or very low message loss. The test environment is designed solely to investigate the near-optimal values for the parameters tested and does not represent how the system will perform outside of the test environment. The latter is investigated in more detail in Section IV-B.

1) Number of transmissions: Figure 9 shows how the message loss is affected by the number of transmissions of the same packet. The requests for ledgers messages are sent in one fragment while the full ledger contains 100 LEs, and with a fragmentation size of 1, it is delivered in 100 fragments. As expected, the message loss decreases as the number of transmissions increases for the full ledgers. However, the message loss for small packets, here represented by request

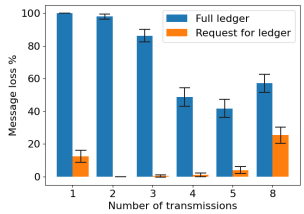
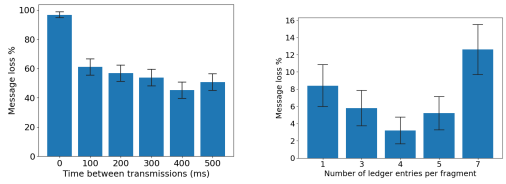


Fig. 9: Number of transmissions.



(a) Time between transmissions. (b) Fragment size.

Fig. 10: Impact on the message loss as a result of varying time between transmissions and fragment size.

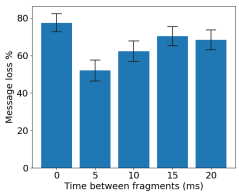
messages for the ledger, increases with a higher number of transmissions. Most of the messages sent in the system, including hashes of ledgers, consist of one packet, so the message loss of small messages is very important, even though the message loss for larger ledgers is also of significance. Choosing 4 transmissions achieves a low message loss of the important small messages while achieving an acceptable message loss of the larger messages, i.e., full ledgers.

2) Time between transmissions: Figure 10(a) shows how the message loss is affected by the time between transmissions. Increasing the time between transmissions will decrease the message loss until 400 ms. That is because what causes packets to be lost in one transmission might be gone before the next one. From 400 ms and upwards, the message loss increases with more time between transmissions. This indicates that the advantage of stretching out the transmissions in time is reduced for values larger than 400 ms.

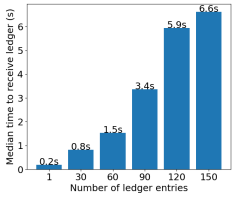
The number of transmissions has to be considered when analyzing the test results. While 400 ms is the optimal time between transmissions for four transmissions, according to these results, that might not be the case for another number of transmissions. With four transmissions, 400 ms between each transmission will stretch the transmission out to 1200 ms in total. The total stretch in time will be lower for fewer transmissions, and the optimal value for the time between transmissions might be higher.

3) Ledger fragment size: Figure 10(b) shows the test results for message loss for different fragment sizes. The figure shows that the message loss is lowest for a fragment size of 4 LEs.

4) Time between fragments: Figure 11(a) shows how the message loss is affected by the time between fragments. As shown, message loss decreases from 0ms to 5ms between



(a) Time between fragments.



(b) Median time to receive full ledgers of length n.

Fig. 11: Impact on the message loss as a result of varying time between fragments and the number of ledger entries' effect on time to receive ledger.

fragments. However, for values higher than 5ms, the message loss increases with the time between fragments. Based on these results, 5ms has been chosen as the optimal value.

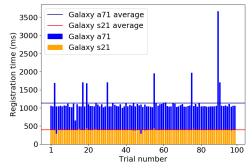
5) Accept ledger timer: The time, β, a user waits before accepting a ledger according to consensus criteria 2 and 3 should be chosen to ensures that most voting messages are received before selecting a ledger. Figure 11(b) shows the median time it takes to receive a full ledger of length n, which is the largest voting message for ledgers of size. n. This value is used to decide β. In reality, the users would receive several hashes in addition to the full ledger. Because all the messages are handled in the same thread on the device, many hashes would increase the time it takes to handle the responses, and should be considered when choosing the value of the timer.

If the value of β is set too low, a ledger might be accepted while a vote is still missing. As all legitimate users will most likely vote for the same ledger, one vote missing will not affect the system for large ledgers. If the value is too large, the user will have to wait longer before they can join the network.

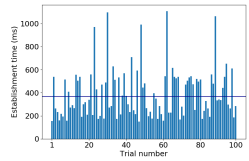
By setting the value of β to 4000ms, full ledgers as large as 90 LEs are most likely to be received and the time is not expected to reduce the user experience.

6) Alone in network timer: The time, α, a new user waits before concluding they are alone in the network should be chosen to give other users time to respond to the new user. As described in Section IV-A1 each packet is sent four times. The time between each transmission is chosen to be 400ms, as also observed in Section IV-A2. Therefore, the total time it takes from the first to the last transmission is 1200ms. That is the case for both the ledger request and its responses. If all but the last transmission is lost, it will take 2400ms plus the transmission time from the user sends the request until it receives a response. Hence, the user should wait 2500ms from the request is sent until the user concludes that they are alone in the network.

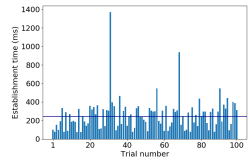
7) Idle time after request: To avoid a DoS attack by request flooding, a user waits γ seconds after responding to a request before the user will respond to new requests. The value for the idle time should be chosen to avoid all transmissions of a request from a legitimate user falling into this window.



(a) Sign up time.



(b) mTLS establishment times.



(c) TCP establishment times.

Fig. 12: Time required to sign up and establish mTLS and TCP connections.

Each request is transmitted four times with 400ms between each transmission, as discussed in Sections IV-A1 and IV-A2. The total time from the first to the last transmission is 900ms. At least two transmissions should always fall outside this window to increase the likelihood of receiving legitimate requests. The idle time must be less than 600ms to ensure that. Therefore the idle time, γ, is chosen to be 590ms.

B. Performance

This section presents the results from testing the applications' performance.

1) Sign up time: To use the application, the user signs up by creating an LE. For users without an existing certificate, this includes checking that the username is available, checking if the device has an Internet connection, and generating keys and certificates.

The test starts when the user has typed in the username and presses the sign-up button and finishes when the LE has been generated. Figure 12(a) shows the results from 100 tests using two different Samsung devices, Galaxy s21 and Galaxy a71.

On average, the Samsung Galaxy s21 used 393ms for sign-up, while the Samsung Galaxy a71 used 1136ms. The worst test runs resulted in test times of over 2s, which is a noticeable amount of time.

The difference in results between the phones is partly due to the difference in clock speed between the phones [16] [17]. The volatility in test times is related to the variability in the time it takes to ping the Google open DNS to check the Internet availability.

2) Connection establishment time: The proof of concept application uses a combination of asymmetric and symmetric encryption to achieve a lower connection establishment time. Two tests have been conducted to find the time it takes to establish a connection, one for each type of connection. The tests start when the user presses the LE of its peer and finish

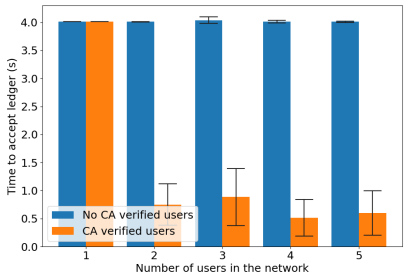


Fig. 13: The time it takes to accept a ledger is affected by the number of peers in the network.

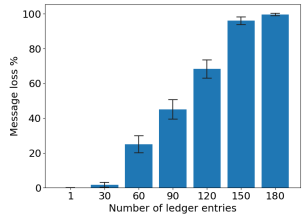


Fig. 14: Effect of ledger size on the message loss.

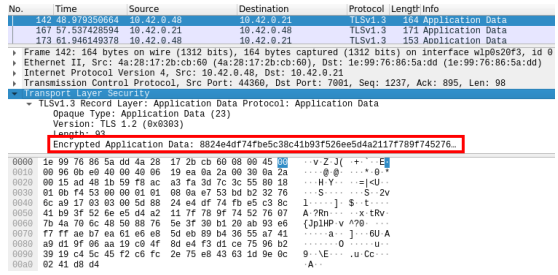


Fig. 15: Wireshark capture of TLS traffic.

when the connection is established. Figure 12(b) shows the connection establishment times for mTLS, and Figure 12(c) shows the connection establishment times for TCP.

The test results show that, on average, it takes 368ms to establish a connection using mTLS and 243ms using TCP. Even though establishing the pure TCP connection takes 34% less time than mTLS, the difference can be negligible.

3) Ledger acceptance: Before a user can sign up for the application, the user has to get the correct ledger or conclude that the user is alone in the network. Therefore, the time for accepting the ledger is an important performance metric. Tests have been conducted to measure this.

The test starts when the user opens the application and finishes when the application has accepted a ledger. The time it takes to accept a ledger depends on the number of users with CA-signed certificates, as described in the ledger acceptance criteria introduced in Section II-B3.

Figure 13 shows how the time to accept the ledger is affected by the number of users in the. Tests have been conducted for a network with CA verified users and in a network without any. The decreased time for more than one CA verified user shows how the first acceptance increases performance.

The probability that the n'th user joining the network accepts a ledger containing the LEs of all the users that have previously joined has been found to be 99% for up to 5 user.

4) Multicast packet loss: A test has been conducted to measure message loss. The test measures how many full ledger messages are lost during a voting. The test is conducted by making one user send a variable-length ledger to another user and recording if the message is lost. The tests have been conducted with the optimal variables found in Section IV-A.

Figure 14 shows that the size of the ledger affects the message loss negatively. Hence, the ledger size affects the performance, as the user is more likely to have to request the full ledger more times when the message loss is high. In particular, we observe that up to 30 ledger entries, i.e., users in the same network, the packet loss is limited to 1.7% which means that the systems enables a fairly robust ledger distribution for up to 30 users, i.e., small to medium-scale networks. Note that although the packet loss increases significantly as the number of users increase, this is not an upper bound on the network scalability as it simply indicates that there is a higher probability that they would be required to forward an additional full ledger request for reaching the consensus.

C. Security

Packets have been captured with and analyzed with Wireshark [18] to verify that the instant messaging traffic is encrypted. A computer was used as an AP to capture packets, and all packets were routed through it and saved. This section presents the security-related findings.

1) Message security: Figure 15 shows a Wireshark capture of a data packet the first time two users communicate. As expected, the packet payload is encrypted using TLS, and the plaintext can not be read without knowledge of the encryption key. In TLSv1.3 the version number gets the default value of 0x0303 - "TLS 1.2", as the version number field is not in use [6]. Hence, Wireshark shows TLS 1.2 in the version field even when TLSv1.3 is used. The protocol column shows the correct protocol, which is TLSv1.3.

Figure 16 shows a Wireshark capture of a data packet the second time two users communicate. The protocol column shows that TCP is used as the transport layer protocol. The payload is encrypted with AES on the application layer, and the ciphertext is shown in the red box in Figure 16. The encrypted payload while using TCP shows that the application works as expected.

The multicast messages sent from the application are sent in cleartext using UDP. Figure 17 shows a multicast message

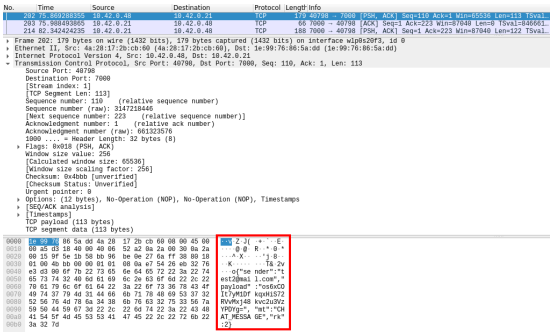


Fig. 16: Wireshark capture of TCP traffic.

broadcasting a LE. The message data is not encrypted and can be read by anyone listening to the multicast group. The message has been signed with the sender's private key.

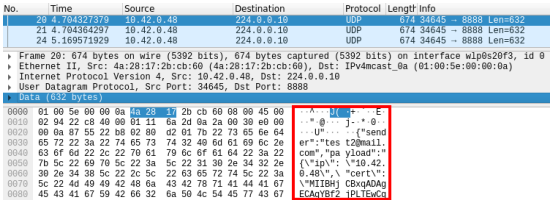


Fig. 17: Wireshark capture of UDP traffic.

V. Conclusion

This paper presents a solution for authentication without a central trusted unit using a distributed authentication scheme and public and private cryptographic keys to provide secure communication. The solution is divided into three independent layers. The first layer presents a solution for setting up communication between the peers relying on Wi-Fi infrastructure. The second layer provides authentication by using a distributed ledger. All users receive the authentication material needed to authenticate their peers. The consensus mechanism ensures that the network as a whole agrees upon a user's identity and thus the user's authentication material. This solution relocates the responsibility of authentication from one single trusted unit to the whole network through cooperation between all network participants. Finally, the third layer provides secure communication relying on mTLS and symmetric encryption. The application's performance has been tested, and its encryption confirmed. In addition, challenges regarding ledger distribution have been identified and addressed through various parameterized mechanisms.

References

[1] I. E. T. F. (IETF). (2018) The transport layer security (TLS) protocol version 1.3. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8446#appendix-C.2

[2] Oracle. (2010) Certificate-based authentication. [Online]. Available: https://docs.oracle.com/cd/E19575-01/820-2765/6nebir7eb/index.html

[3] "IEEE standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pp. 1–4379, 2021.

[4] A. Sunyaev, *Distributed Ledger Technology*. Cham: Springer International Publishing, 2020, pp. 265–299. [Online]. Available: https://doi.org/10.1007/978-3-030-34957-8_9

[5] F. Saleh, "Blockchain without Waste: Proof-of-Stake," *The Review of Financial Studies*, vol. 34, no. 3, pp. 1156–1190, 07 2020. [Online]. Available: https://doi.org/10.1093/rfs/hhaa075

[6] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," Internet Requests for Comments, IETF, RFC 8446, 8 2018. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8446

[7] Ø. Sigholt, B. Tola, and Y. Jiang, "Keeping connected when the mobile social network goes offline," in *2019 International conference on wireless and mobile computing, networking and communications (WiMob)*. IEEE, 2019, pp. 59–64.

[8] K. L. Skaug, E. B. Smebye, B. Tola, and Y. Jiang, "Keeping connected in internet-isolated locations," in *2022 Seventh International Conference On Mobile And Secure Services (MobiSecServ)*. IEEE, 2022, pp. 1–7.

[9] M. T. Hammi, B. Hammi, P. Bellot, and A. Serhrouchni, "Bubbles of trust: A decentralized blockchain-based authentication system for iot," *Computers & Security*, vol. 78, pp. 126–142, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818300890

[10] U. Maurer and S. Wolf, "The diffie–hellman protocol," in *Designs, Codes and Cryptography 19*, 2000, p. 147–171.

[11] W. Stallings, *Cryptography and Network Security: Principles and Practice, Global Edition*. Pearson Education Limited, 2016.

[12] M. Marlinspike and T. Perrin, "The double ratchet algorithm," Open Whisper System, Tech. Rep., Nov. 2016.

[13] D. Cooper, S. Santesson, S. Farrel, S. Boeyen, R. Housley, and T. Polk, "Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile," Internet Requests for Comments, IETF, RFC 5280, 5 2008. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5280

[14] S. Aggarwal and N. Kumar, "Chapter twenty - attacks on blockchain," in *The Blockchain Technology for Secure and Smart Applications across Industry Verticals*, ser. Advances in Computers, S. Aggarwal, N. Kumar, and P. Raj, Eds. Elsevier, 2021, vol. 121, pp. 399–410. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245820300759

[15] J. Korhonen and Y. Wang, "Effect of packet size on loss rate and delay in wireless links," in *IEEE Wireless Communications and Networking Conference, 2005*, vol. 3, 2005, pp. 1608–1613 Vol. 3.

[16] L. SAMSUNG ELECTRONICS CO. (2022) Specifications galaxy s21 fe — s21 — s21+ 5g. [Online]. Available: https://www.samsung.com/no/smartphones/galaxy-s21-5g/specs/

[17] ——. (2022) Specifications galaxy s21 fe — s21 — s21+ 5g. [Online]. Available: https://www.samsung.com/no/business/smartphones/galaxy-a/galaxy-a71-a715-sm-a715fzkunee/

[18] Wireshark. (2022) Wireshark. [Online]. Available: https://www.wireshark.org/

NTNU
Norwegian University of
Science and Technology