

ANDREA ERINA KOMO

An efficient method to provide auditable messages  
exchanged in instant messaging applications

Corrected Version

São Paulo  
2023

ANDREA ERINA KOMO

**An efficient method to provide auditable messages  
exchanged in instant messaging applications**

**Corrected Version**

Master Dissertation presented to the Departamento de Engenharia de Computação e Sistemas Digitais at the Escola Politécnica, Universidade de São Paulo, Brazil to obtain the degree of Master of Science.

São Paulo  
2023

ANDREA ERINA KOMO

**An efficient method to provide auditable messages  
exchanged in instant messaging applications**

**Corrected Version**

Master Dissertation presented to the Departamento de Engenharia de Computação e Sistemas Digitais at the Escola Politécnica, Universidade de São Paulo, Brazil to obtain the degree of Master of Science.

Concentration area:

Computer Engineering

Supervisor:

Prof. Dr. Marcos Antonio Simplicio Junior

São Paulo  
2023

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 17 de Janeiro de 2023

Assinatura do autor: Andrea Erina Komo

Assinatura do orientador: Markus

#### Catálogo-na-publicação

Komo, Andrea Erina

An efficient method to provide auditable messages exchanged in instant messaging applications / A. E. Komo – versão corr. – São Paulo, 2023.

92 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Sistema distribuído 2.Segurança da informação 3.Hash chain 4.Aplicativo celular 5.Troca de mensagens I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

KOMO, Andrea Erina. *An efficient method to provide auditable messages exchanged in instant messaging applications*. 2023. Dissertação (Mestrado em Ciências) - Versão corrigida – Escola Politécnica, Universidade de São Paulo, São Paulo, 2023.

Aprovado em: 29/11/2022

Banca Examinadora

Prof. Dr. Marcos Antonio Simplicio Junior  
Instituição: Escola Politécnica da Universidade de São Paulo  
Julgamento: Aprovado

Prof. Dr. Charles Christian Miers  
Instituição: Universidade do Estado de Santa Catarina  
Julgamento: Aprovado

Prof. Dr. Carlos Eduardo da Silva  
Instituição: Sheffield Hallam University  
Julgamento: Aprovado

# ACKNOWLEDGMENTS

To Prof. Dr. Marcos A. Simplicio Jr. and Prof. Dra. Cíntia B. Margi, for teaching me a lot, contributing to my scientific and intellectual growth, and for support during this work process.

To the Polytechnic School from USP and the Department of Computational and Digital Systems Engineering (PCS), for the opportunity to carry out the master's course.

To my family and friends, for their presence and all the support and encouragement they gave me in difficult times.

A special thanks to Felipe Valencia de Almeida who has always accompanied me from graduation to our entry into the master's program; to my seniors Jefferson Ricardini, Marcos Vinicius Maciel da Silva, Eduardo Lopes Cominetti, and Renan Cerqueira Afonso Alves who helped me in this master's course; to Eduardo Barasal Morales who always encouraged me not to give up regardless of the situation; to Felipe Brigatto who helped me in the development and tests of the PoC prototypes.

This study was financed in part by LG Electronics and by Ripple.

# RESUMO

KOMO, Andrea Erina. *An efficient method to provide auditable messages exchanged in instant messaging applications*. 2023. Dissertação (Mestrado em Ciências e Engenharia de Computação) - Versão corrigida – Escola Politécnica, Universidade de São Paulo, São Paulo, 2023.

A integridade e a autenticidade de mensagens em aplicativos de comunicação móvel tornou-se uma questão importante, uma vez que esses aplicativos estão sendo usados como ferramentas corporativas. Em particular, não é incomum que as mensagens trocadas sejam usadas como documentos de negociação, comprovantes de vendas, ou transações bancárias. No entanto, por projeto, a maioria desses aplicativos não fornece qualquer recurso de verificação para confirmar a integridade ou autenticidade das conversas após estas serem entregues aos seus destinos. Na verdade, existem vários exemplos na literatura de como é possível modificar de forma imperceptível registros em aplicativos como WhatsApp, Telegram e outros. Além disso, a maioria dos aplicativos disponíveis no mercado utiliza uma arquitetura centralizada, que pode levantar dúvidas sobre a integridade e autenticidade das mensagens. Afinal, como a entidade central controla o serviço, esta tem a liberdade técnica para executar um ataque do “homem no meio” (MitM) e talvez modificar mensagens sem ser detectada. Todas essas características podem comprometer as negociações feitas nessas plataformas de troca de mensagens. Para resolver esta questão da integridade das conversas, este trabalho propõe uma estrutura de mensagens composta por *hashes* criptográficos encadeados (ou *hash chain*), garantindo assim uma forma de auditar e verificar as conversas. Além disso, é proposta a utilização de um sistema distribuído como infraestrutura de comunicação, levando a um cenário mais independente de qualquer entidade controladora que possa interferir no fluxo de mensagens. Para isso, o sistema proposto combina tecnologias como PGP (*Pretty Good Privacy*), comunicações *peer-to-peer* (P2P) e um mecanismo de encadeamento de *hashes* com privacidade seletiva. Os diferentes módulos que compõem a solução são independente de arquitetura e, portanto, podem ser integrados de forma individual a qualquer aplicativo de mensagem instantânea. Os testes documentados neste trabalho mostraram-se satisfatórios em termos de tempos de execução inferiores a dois segundos e robustez da solução de auditoria independente.

**Palavras-Chave** – Sistema distribuído. Segurança da informação. *Hash chain*. Aplicativo celular. Troca de mensagens.

# ABSTRACT

KOMO, A. E. An efficient method to provide auditable messages exchanged in instant messaging applications. 2023. Dissertation (Master of Science) - Corrected version – Escola Politécnica, Universidade de São Paulo, São Paulo, 2023.

Message integrity in mobile communication apps has become an important issue since such apps are being used as corporate tools. In particular, it is not uncommon for messages thereby exchanged to be used as negotiation documents, sales, bank transactions. However, by design, most of these applications do not provide any verification feature to confirm the integrity or authenticity of conversations after they have been delivered to their destinations. There are several examples in the literature of how it is possible to imperceptibly modify records in apps such as WhatsApp, Telegram, and others. In addition, most applications available on the market use a centralized architecture, which may raise doubts about messages' integrity and authenticity. After all, because the central entity controls the service, it has the technical freedom to execute a man-in-the-middle (MitM) attack and perhaps modify messages without being detected. All of these characteristics can compromise the negotiations made on these message exchange platforms. To resolve this issue of conversation integrity, this work proposes a message structure composed of chained cryptographic hashes (hash chain) that ensure a way to audit and verify conversations. Also, we propose to use a distributed system as a communication infrastructure, leading to a scenario that is more independent from any controlling entity that may interfere with the flow of messages. For this, the proposed system combines technologies such as PGP (Pretty Good Privacy), peer-to-peer (P2P) communications, and a hash-chaining mechanism with selective disclosure. The different modules that make up this solution are architecture-independent and, therefore, can be integrated individually into any instant messaging application. The tests documented in this work proved to be satisfactory in terms of execution times of less than two seconds and the robustness of the independent audit solution.

**Keywords** – Distributed systems. Security. Hash chain. Mobile application. Messages exchange.



# LIST OF FIGURES

1	Sequence Diagram of Client–Server structure . . . . .	16
2	Denial-of-Service attack . . . . .	17
3	Sequence Diagram of Man-in-the-Middle attack . . . . .	18
4	Representation of the hash chain structure . . . . .	23
5	Asymmetric-key cryptography: encryption . . . . .	24
6	Asymmetric-key cryptography: digital signature . . . . .	24
7	PGP web of trust . . . . .	26
8	Example of conversation with deleted message . . . . .	33
9	Example of conversation with edited message . . . . .	34
10	Sequence diagram of proposed app’s connection architecture . . . . .	47
11	Hash chain structure for messages. Assuming that user 1 sent messages $\mathcal{N}$ and $\mathcal{N}+1$ , only his last signature needs to be stored . . . . .	48
12	Sequence diagram of proposed app’s conversation dynamics . . . . .	50
13	Messages chat hash chain . . . . .	51
14	Conversation with an edited message . . . . .	52
15	Conversation with messages’ order changed . . . . .	52
16	Conversation with deleted message . . . . .	53
17	Conversation with an inserted message . . . . .	53
18	Example of message blocks’ collision . . . . .	54
19	State machine for handling message blocks’ collision . . . . .	55
20	Screenshot of the app developed by [ARAKAKI; LEVY, 2017] . . . . .	57
21	Screenshot of PoC Android . . . . .	58
22	Screenshot of PoC Python . . . . .	59
23	Chat generated in the Python PoC . . . . .	70

24	Hash chain generated in the Python PoC . . . . .	71
25	Auditing the chat with correct messages . . . . .	71
26	Auditing the chat with a different message . . . . .	72
27	Hash chain modified: changing the fourth and the fifth registers . . . . .	72
28	Auditing the chat with message order changed . . . . .	73
29	Hash chain modified: deleting the fourth register . . . . .	73
30	Auditing the chat with message deleted . . . . .	74
31	Hash chain modified: adding a new the fourth register . . . . .	74
32	Auditing the chat with message added . . . . .	75
33	Auditing the full chat with correct messages' content . . . . .	76
34	Chat generated with collision in the Python PoC . . . . .	77
35	Hash chain generated from chat with collision in the Python PoC . . . . .	78
36	Partial audit tests from chat with collision . . . . .	79
37	Partial audit test starting after collision . . . . .	79
38	Auditing full chat with collision . . . . .	80

# LIST OF TABLES

1	Example of conversation with modified message sequence . . . . .	19
2	Comparing IM apps structure . . . . .	36
3	Comparing IM apps confidentiality and privacy mechanisms . . . . .	36
4	Comparing IM apps audit mechanisms . . . . .	37
5	IM apps others security features . . . . .	37
6	Information stored in the DHT . . . . .	45
7	Specification of the test devices . . . . .	62
8	ECDSA signature benchmark . . . . .	63
9	ECDSA verification benchmark . . . . .	63
10	ECDH key exchange benchmark . . . . .	63
11	SHA3-256 benchmark mean times (ms) . . . . .	64
12	SHA3-256 benchmark standard deviation times (ms) . . . . .	64
13	SHA3-256 benchmark median times (ms) . . . . .	65
14	PGP signature benchmark . . . . .	65
15	PGP encryption benchmark in Samsung Galaxy J2 Prime . . . . .	66
16	PGP encryption benchmark in How HT-705 tablet . . . . .	66
17	PGP encryption benchmark in Xiaomi Redmi Note 7 . . . . .	66
18	PGP encryption benchmark in Samsung Galaxy A22 . . . . .	67
19	PGP decryption benchmark in Samsung Galaxy J2 Prime . . . . .	67
20	PGP decryption benchmark in How HT-705 tablet . . . . .	68
21	PGP decryption benchmark in Xiaomi Redmi Note 7 . . . . .	68
22	PGP decryption benchmark in Samsung Galaxy A22 . . . . .	68
23	SHA3-256 benchmark . . . . .	90

24	PGP encryption benchmark . . . . .	91
25	PGP decryption benchmark . . . . .	92

# LIST OF ABBREVIATIONS AND ACRONYMS

app	application
AE	Authenticated Encryption
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AtD	Agree-to-Disagree
BHP	Bramble Handshake Protocol
BQP	Bramble QR Code Protocol
CBC	Cipher Block Chaining
CPU	Central Processing Unit
DH	Diffie–Hellman
DHT	Distributed Hash Table
DoS	Denial-of-Service
DDoS	Distributed Denial-of-Service
DSA	Digital Signature Algorithm
EC	Elliptic Curve
E2E	End-to-end
GCM	Galois/Counter Mode
GDPR	General Data Protection Regulation
HMAC	Hash-based Message Authentication Code
ID	Identity
IGE	Infinite Garble Extension
IM	Instant messaging
IoT	Internet of Things
IP	Internet Protocol

IV	Initialization Vector
LAN	Local Area Network
LGPD	<i>Lei Geral de Proteção de Dados Pessoais</i>
MAC	Messages Authentication Code
MitM	Man-in-the-Middle
NAT	Network Address Translation
P2P	Peer-to-Peer
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
PoC	Proof of Concept
PRF	Pseudorandom Function
RAM	Random-access memory
SFS	Self-certifying File System
SHA	Secure Hash Algorithm
SMS	Short Message Service
SPOF	Single Point of Failure
TLS	Transport Layer Security
Tor	The Onion Router

# CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>15</b>
1.1	Motivation . . . . .	15
1.2	Goals . . . . .	20
1.3	Contributions . . . . .	20
1.4	Outline . . . . .	21
<b>2</b>	<b>BASIC CONCEPTS AND NOTATION</b>	<b>22</b>
2.1	Cryptographic Hash Functions . . . . .	22
2.2	Hash Chain . . . . .	23
2.3	Asymmetric-key cryptography . . . . .	23
2.4	Diffie–Hellman key exchange . . . . .	25
2.5	Pretty Good Privacy (PGP) . . . . .	26
2.6	Distributed Hash Table (DHT) . . . . .	27
<b>3</b>	<b>SPECIFICATION OF PROJECT REQUIREMENTS</b>	<b>28</b>
3.1	Scope . . . . .	28
3.2	Requirements . . . . .	29
3.2.1	Functional Requirements . . . . .	29
3.2.2	Non-Functional Requirements . . . . .	30
3.3	Application scenario and target audience . . . . .	31
<b>4</b>	<b>RELATED WORKS</b>	<b>32</b>
<b>5</b>	<b>METHODS AND MATERIALS</b>	<b>39</b>
5.1	Use cases . . . . .	39

5.1.1	Use case: Register new user . . . . .	39
5.1.2	Use case: Send a message . . . . .	40
5.1.3	Use case: Trust another user . . . . .	42
5.1.4	Use case: Audit a chat . . . . .	43
<b>6</b>	<b>PROPOSED ARCHITECTURE</b>	<b>44</b>
6.1	Connection network . . . . .	44
6.2	Security structure . . . . .	47
6.2.1	Analysis of the records' integrity with the proposed hash chain . . .	51
6.2.2	Collision case: Agree-to-Disagree algorithm . . . . .	54
<b>7</b>	<b>PROOF OF CONCEPT</b>	<b>56</b>
7.1	Android app PoC . . . . .	56
7.2	Python chat PoC . . . . .	58
<b>8</b>	<b>TESTS AND VALIDATIONS</b>	<b>60</b>
8.1	Functional Validation Tests . . . . .	60
8.1.1	Android PoC . . . . .	60
8.1.2	Python PoC . . . . .	61
8.2	Efficiency analysis: Android benchmarks . . . . .	62
8.3	Integrity and auditing testing of conversations . . . . .	69
<b>9</b>	<b>CONCLUSION</b>	<b>81</b>
9.1	Publications . . . . .	82
9.2	Future Works . . . . .	82
	<b>REFERENCES</b>	<b>84</b>
	<b>APPENDIX A – Benchmark tables</b>	<b>89</b>



# 1 INTRODUCTION

Seeing the development of computing, Mark Weiser proposed that one-day computer technologies would be incorporated everywhere in our lives. With this, he created the ubiquitous computing concept [WEISER, 1999]. Time has passed and the world is more like Weiser predicted. The technological development enabled industries automation, autonomous cars, smartphones, smart homes, and Internet of Things (IoT) devices.

Among all these technologies and devices, an essential one in the 21st century is the smartphone and its applications, having over 6.3 billion users in 2021<sup>1</sup>. Instant messaging applications (IM apps) have changed the way people interact and communicate. These apps allow daily chat with friends and family, as well allow to do bank transactions, negotiations, sales, and business chats, being very convenient. Some famous apps like Facebook’s Messenger have more than 1.3 billion users in 2017 [FACEBOOK, 2017], WhatsApp reached 2 billion users around the world in 2020 [WHATSAPP, 2020], and Telegram surpassed 500 million monthly active users in 2021 [DUROV, 2021]. These systems are very user-friendly and became part of people’s lives, but we have inquiry if those systems are secure for enterprise use. This work will focus on smartphone apps and security, we will analyze the security of some different types of IM apps systems existents and propose a solution that improves chat integrity and users’ authenticity in these systems.

## 1.1 Motivation

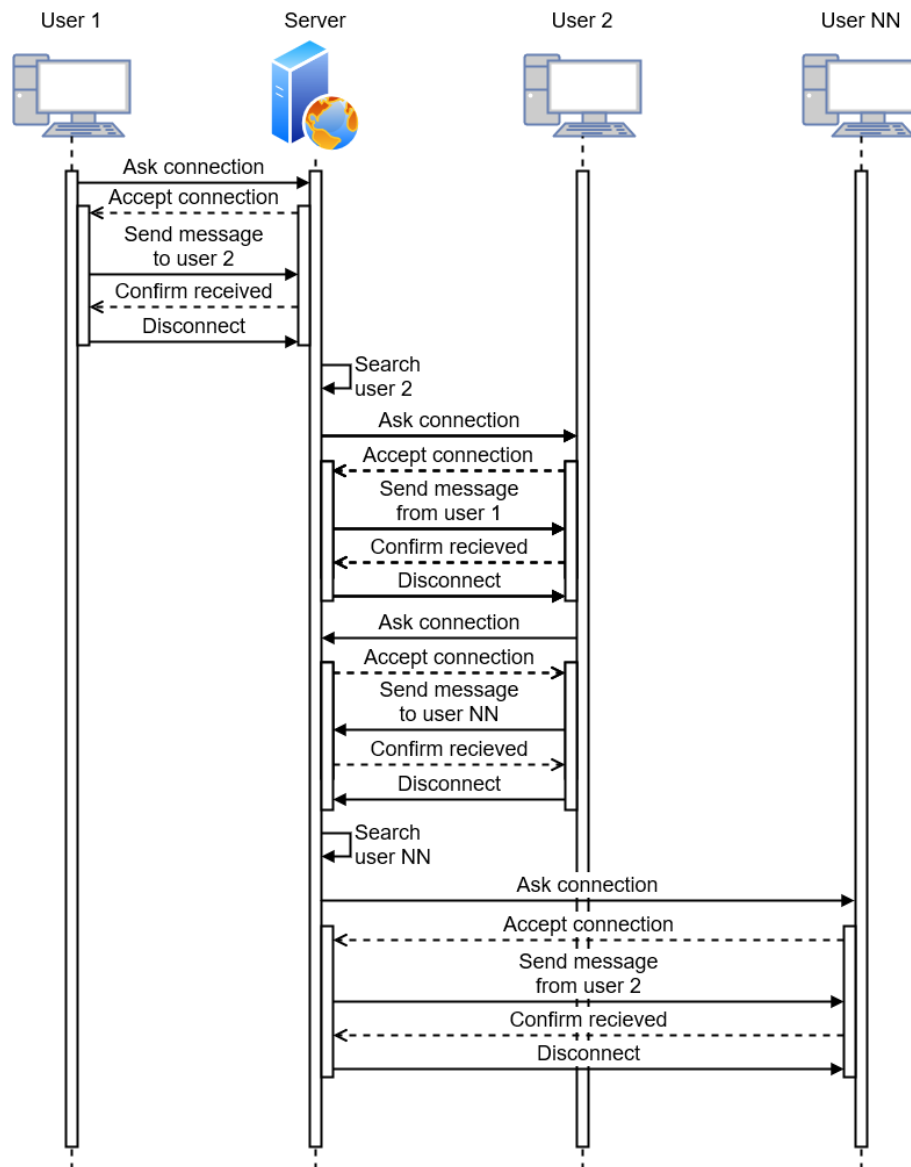
Instant messaging applications available on the market are systems that provide practical and efficient communication service. Most of these systems use client–server connection structure [KUROSE; ROSS, 2013] that each system has a server to provide users (clients) communication. If a user  $U_1$  wants to send a message  $M$  to another user  $U_2$ , first  $U_1$  sends  $M$  to server  $S$ , then this server searches the connection information from  $U_2$  and finally  $S$  sends  $M$  to  $U_2$ . The communication between  $U_1$ ,  $U_2$ , and any other users

---

<sup>1</sup>Source: <<https://www.bankmycell.com/blog/how-many-phones-are-in-the-world>>

depends fundamentally on the server  $S$  and therefore this type of approach is considered centralized in the server as shown in Figure 1.

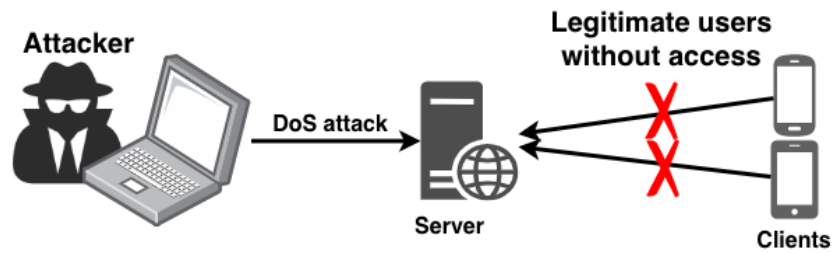
Figure 1: Sequence Diagram of Client–Server structure



Source: Authors.

Centralized structures may present a Single Point of Failure (SPOF), if this point of the system fails, then all system stop working. For this reason, attackers usually see SPOF as a security vulnerability and target it, for example, doing a Denial-of-Service (DoS) attack [BROWN; STALLINGS, 2015] in a centralized server affecting the system's availability. In Figure 2, a DoS attack target the server and if successful then legitimate users are no longer able to use the service during the downtime.

Figure 2: Denial-of-Service attack



Source: Authors.

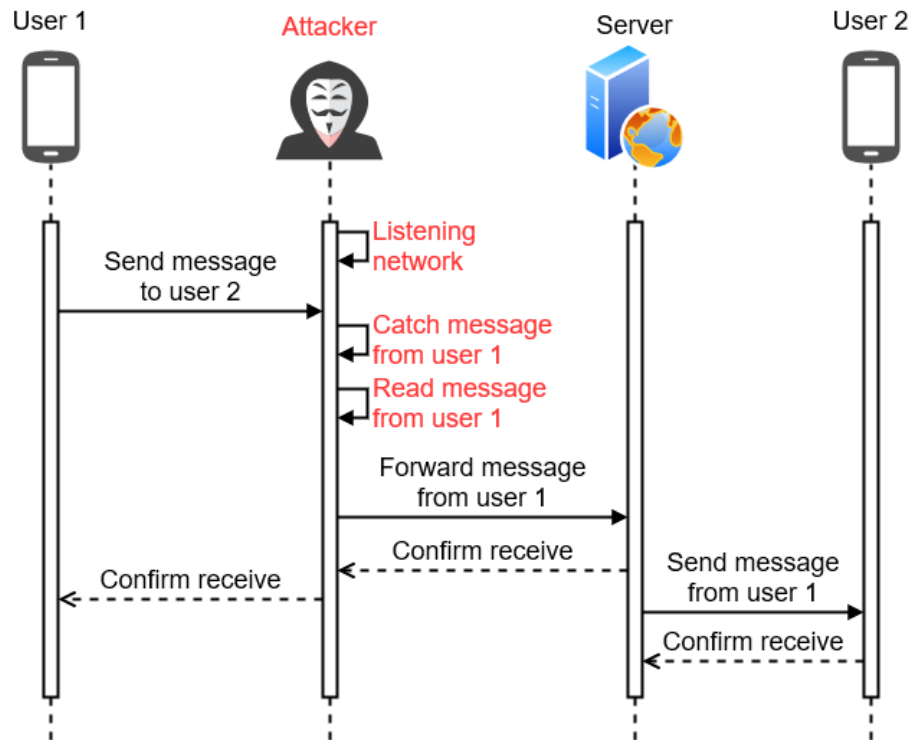
There are types of DoS attacks that can cause this unavailability:

- Vulnerability DoS: When the attacker exploits a well-known vulnerability of the system to slow or crash the service.
- Flooding DoS: When the attacker generates a flood of requests that overwork the system until the service is unable to respond anymore.
  - Distributed Denial-of-Service (DDoS): When the attacker generates a flood of requests using a “zombie” botnet, a group of distributed computers that have been infected by malware and have come under the control of the attacker.

The temporary judicial blockage of the WhatsApp application in Brazil in 2015 and 2016 [BARRETO; LIMA, 2016] can be seen as the DoS attack because this shut down all communication of users over the app.

Another centralization vulnerability is based on the premise that the entities responsible for the services are reliable. Because this entity is the server controller, it can technically carry out a Man-in-the-Middle (MitM) attack [MEYER; WETZEL, 2004]. In a MitM attack, the attacker is in the middle of the communication capturing information, as illustrated in Figure 3, this mainly affects users’ confidentiality and privacy. This type of attack can be performed by any attacker, however, in a centralized system, it is much easier to make a MitM attack if you are the system’s insider.

Figure 3: Sequence Diagram of Man-in-the-Middle attack



Source: Authors.

The case Facebook–Cambridge Analytica [MEREDITH, 2018] had repercussions in the press because the entities had performed a MitM attack in this case. It was found that Facebook improperly provided data from its users to Cambridge Analytica, a data analysis company, showing that the service provider company is not always reliable. After this case, the governments have been concerned more about citizens' privacy and created data protection laws, such as General Data Protection Regulation (GDPR) in the European Union [THE EUROPEAN PARLIAMENT; THE COUNCIL OF THE EUROPEAN UNION, 2016] and Brazil's law number 13.709, the General Law on Protection of Personal Data (LGPD) [BRASIL, 2018]. These laws are important and have changed the way data are used and stored, including in IM apps.

The vulnerabilities of applications available on the market are not limited to centralized architecture. In a general context, the following security vulnerabilities can also be observed:

- Lack of confidentiality: Some applications do not have end-to-end encryption mechanisms by default for the messages exchanged [TELEGRAM, 2018]. This can be a problem for companies that use these services to exchange sensitive information

because it makes it easier to obtain sensitive data through MitM.

- **Verification of authenticity:** Most applications use unreliable methods to verify the authenticity of users mainly during initial registration on their systems [SCHRIETWIESER et al., 2012]. In this way, it is necessary to have the active participation of each user to verify the interlocutors’ authenticity. Unfortunately, many users are unaware of the risks and potential damage resulting from the exposure of their personal information, so few are careful to do this type of security check [ABU-SALMA et al., 2017].
- **Integrity and non-repudiation of communication:** If the messages exchanged in some applications can be used as judicial evidence [BRASIL, 2017], it is important to prove the integrity and non-repudiation of messages and conversation [SCHLIEP; HOPPER, 2018]. Not all applications ensure that messages have not been manufactured or modified and that the conversation has not undergone any changes in content or order. In a hypothetical situation, if the messages are stored in a central database and an attacker modifies the records, false evidence can be created. For example, in Table 1, on the left, we have the authentic chat showing the legitimate messages’ sequence. On the right, we have a modified chat where the messages “Twice a week” and “No” from  $\mathcal{B}$  changed order. As we can observe, just this modification changed all the chat’s meaning.

Table 1: Example of conversation with modified message sequence

<b>Authentic chat</b>	<b>Modified chat (change order)</b>
$\mathcal{A}$ : Do you go to the gym?	$\mathcal{A}$ : Do you go to the gym?
$\mathcal{B}$ : Twice a week.	$\mathcal{B}$ : No.
$\mathcal{A}$ : Do you smoke?	$\mathcal{A}$ : Do you smoke?
$\mathcal{B}$ : No.	$\mathcal{B}$ : Twice a week.

Source: Authors.

Even with security vulnerabilities, it is noted that apps have gained greater integration in the corporate world. In January 2021, BMW Group from Brazil launched a fast answering service using the Whatsapp application, offering the consumer the opportunity to ask questions, stay on top of the news, learn about products, check special conditions, and request proposals [CICHINI, 2021]. In May 2021, Whatsapp allowed payment function in its IM app system for Brazilians [WHATSAPP, 2021]. The justifications for these

integrations are popularity, availability, and efficiency of the apps, but security is a system characteristic that tends to conflict with others [ELAHI; YU, 2007]. As such, security often ends up being postponed over usability and performance at systems development.

## 1.2 Goals

Considering the scenarios described in Section 1.1, this work proposes a solution that mitigates vulnerabilities and contributes to the security of users and corporations that use popular instant messaging apps. Thus, the goals of this research are to develop a communication architecture that is secure and allows to audit the conversations carried out by smartphones' IM apps. In particular, it is expected to add the characteristics of integrity, authenticity, and non-repudiation to the exchanged messages in order to ensure that the conversations can undoubtedly be audited and validated aiming mainly at the scenarios of negotiations carried out on the IM platforms. For this, we intend to use a decentralized instant messaging application [ARAKAKI; LEVY, 2017] and incorporate some cryptography mechanisms such as public keys and hash chain to allow reliable audits of the conversations.

## 1.3 Contributions

As we have seen in Section 1.1, there are some vulnerabilities and points for improvement in the security of instant messaging applications. One of the proposed contributions is to change this systems architecture to a distributed structure, using peer-to-peer (P2P) at the connections, then it becomes more independent, resilient, and secure since the service is maintained by the network users themselves [KOMO et al., 2018]. However, just distributing the system does not improve the system's auditability.

Most instant communication solutions focus on plausible deniability [NELSON; ASKAROV, 2022], rather than providing auditability among its security services; hence, even in such a centralized scenario, auditing messages would still not be an easy task (if at all possible). Then, as a second contribution, we propose a combination of security techniques in order to obtain integrity, authenticity, and non-repudiation in any instant messaging app system, whether centralized or distributed. In specific we will use a cryptographic hash function to create a hash chain, this will generate secure, verifiable, orderly records, and selective disclosure of messages. An example of a distributed system that uses this technique to maintain record integrity is the cryptocurrency Bitcoin [NAKAMOTO, 2008]

and the version control system Git<sup>2</sup>. Furthermore, we will use PGP digital signatures to ensure the system's users' authenticity and non-repudiation. The first and second contributions are independent and, therefore, can be integrated individually into any instant messaging application.

## 1.4 Outline

The remainder of this document is organized as follows. Chapter 2 presents the cryptographic concepts and notations used in the entire document. Chapter 3 defines the scope of this project and the requirements that must be fulfilled in the work. Chapter 4 comments on the related works present in the literature. Chapter 5 specifies the use cases necessary to reach this work's goals. Chapter 6 details the design of the proposed architecture based on the use cases. Chapter 7 describes a proof of concept (PoC) prototype implemented. Chapter 8 presents tests results and analyses done with the PoC. Finally, we conclude in Chapter 9 showing final considerations, publications, and future works.

---

<sup>2</sup><https://git-scm.com/doc>

## 2 BASIC CONCEPTS AND NOTATION

In this chapter, we review the building blocks and notation applied in the construction of our proposed architecture further detailed in Chapter 6. Sections 2.1 to 2.5 explain security concepts such as cryptographic hash functions, asymmetric-key cryptography, Diffie–Hellman, and PGP. Section 2.6 present Distributed Hash Table (DHT) used at network connection. We note that all definitions are quite standard and a reader with a background in these concepts may skip the following sections.

### 2.1 Cryptographic Hash Functions

Hash function is a one-way function that can be referred to as a “summary function”. Given a message  $M$ , the hash function returns a “summary” (or hash)  $H$  from this message  $M$  [STALLINGS, 2014]. Hash function presents the following properties:

- Compression: Regardless of the message  $M$  size, the function will return a fixed and specific size hash  $H$ .
- Ease of computation: The computational cost of this function is extremely low. If you have the message  $M$ , it is very easy to calculate your hash  $H$ .

Cryptographic hash function has some other properties:

- (First) Preimage resistance: Given the hash  $H$ , it must be computationally infeasible to find the message  $M$  that generates  $H$ .
- Second preimage resistance: Given a message-hash pair  $(M, H)$ , it must be computationally infeasible to find another message  $M'$  distinct from  $M$  that generates the same hash  $H$ .
- Collision resistance: It must be computationally infeasible to find two different messages  $M$  and  $M'$  that generate the same hash  $H$ .

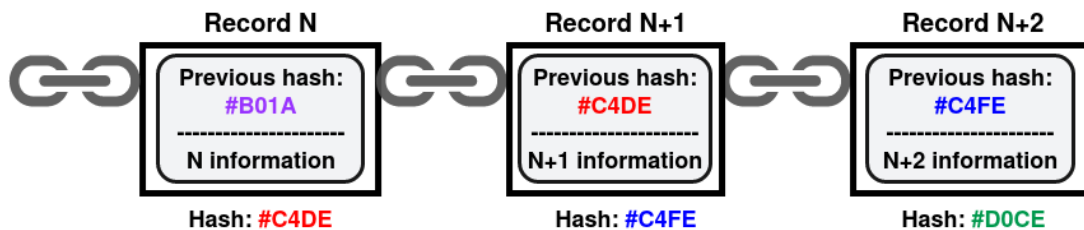


Note that the hash depends only on the input data and any minimal changes to it will result in a completely different output hash. In addition, cryptographic hash functions do not have secret cryptographic keys, so anyone with the correct input data can generate the hash. All these characteristics make the cryptographic hash easy to verify, but difficult to discover the input data.

## 2.2 Hash Chain

Hash chain is a structure built to ensure the ordering and integrity of a set of information. Considering a hash chain with  $N$  registered information, if we want to insert some new information  $N + 1$ , we will need to calculate the cryptographic hash of the previous message  $N$  and save this hash together with the new information at the new record of the chain as illustrated in Figure 4.

Figure 4: Representation of the hash chain structure



Source: Authors.

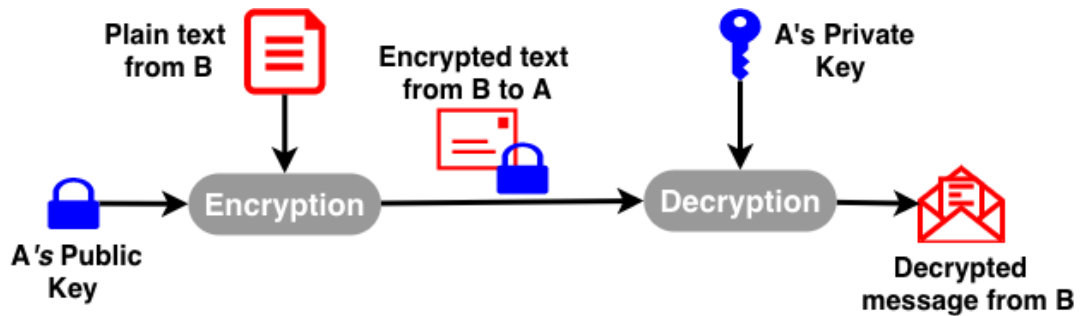
As mentioned in Section 2.1, any change to the input data of a cryptographic hash function changes the output hash. Thus, the integrity of each record and the sequence is guaranteed, because a record depends on these previous records by a cryptographic hash.

## 2.3 Asymmetric-key cryptography

Also known as public-key cryptography [STALLINGS, 2014], algorithms of this type have this name because they use two distinct cryptographic keys, one public and one private. As the name says, all users have a public key that should be widely known by other users and a secret private key is known exclusively by its owner. A feature of these keys is that they are logically linked by generation, but it is computationally infeasible to discover the private key by analyzing its corresponding public key.

Depending on the key used in the process, the algorithms may perform some actions. Using the user  $\mathcal{A}$ 's public key with the algorithms, a user  $\mathcal{B}$  can **encrypt** information and then send a confidential message directed to  $\mathcal{A}$ . In encrypting information with a public key, only the correlate private key can decrypt it, as shown in Figure 5.

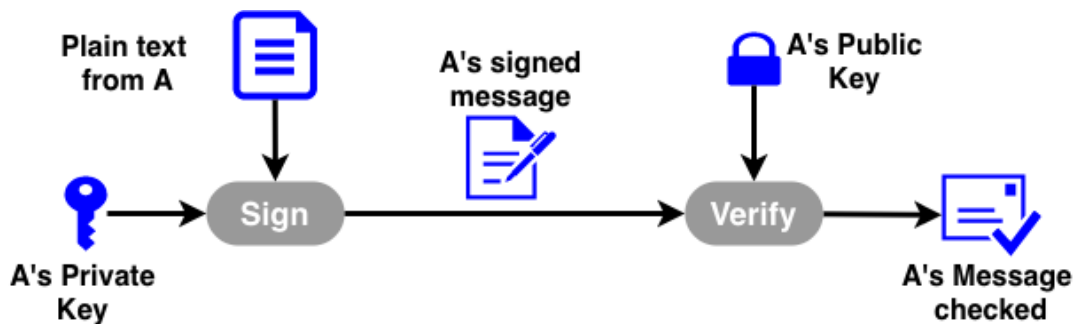
Figure 5: Asymmetric-key cryptography: encryption



Source: Authors.

The reverse process can also be done and it is known as **digital signature**. In this process, the user  $\mathcal{A}$  uses his private key in encryption algorithms, and using his correlated public key with the decryption algorithm is possible to verify this signature, as shown in Figure 6. As everyone can decrypt the information, everyone can check that it was unquestionably generated by the owner of the private key, thus ensuring the integrity, authenticity, and non-repudiation of the information.

Figure 6: Asymmetric-key cryptography: digital signature



Source: Authors.

In this context, another important concept is that of **digital certificate**. This certificate is an electronic public document that contains information about a public key and its owner. Based on this document, it is possible to verify the digital signature and link it to a specific user, so that no one, not even the owner of the key, can deny the authenticity of

the signature. Therefore, it is important to correctly generate and propagate certificates among users. An example of a digital certificate is the X.509 standard that is used at sites with HTTPS protocol.

## 2.4 Diffie–Hellman key exchange

Diffie–Hellman (DH) key exchange [DIFFIE; HELLMAN, 1976] is a method created to allow a key agreement over a public network. This method ensures the secure exchange of information based on **discrete logarithm** problems that present the following properties:

- Given  $a$ ,  $p$  and  $x$ , it is computationally “easy” to calculate  $y = a^x \bmod p$ .
- Given  $a$ ,  $p$  and  $y$ , it is computationally “hard” to calculate  $x$ .

In the Diffie-Hellman key exchange algorithm, the two parties’ users establish an equal secret using asymmetric-key pair and the discrete logarithm problem. This shared key is generated using the following parameters:

- $q$ : a prime number;
- $p$ : a prime number such that  $p - 1$  is a multiple of  $q$ ;
- $g$ : a integer between 2 and  $p - 2$  such that  $g^q \bmod p = 1$
- $r$ : the private key that is an aleatory number between 1 and  $q - 1$ ;
- $u$ : the public key that is  $u = g^r \bmod p$ ;

The user  $\mathcal{A}$  will calculate  $K = (u_B)^{r_A} \bmod p$ . And the user  $\mathcal{B}$  will calculate  $K = (u_A)^{r_B} \bmod p$ . This shared secret  $K$  can be used in symmetric-key cryptography for secret communication between the parts and to ensure privacy. Note that these two calculations produce identical results by the rules of modular arithmetic.

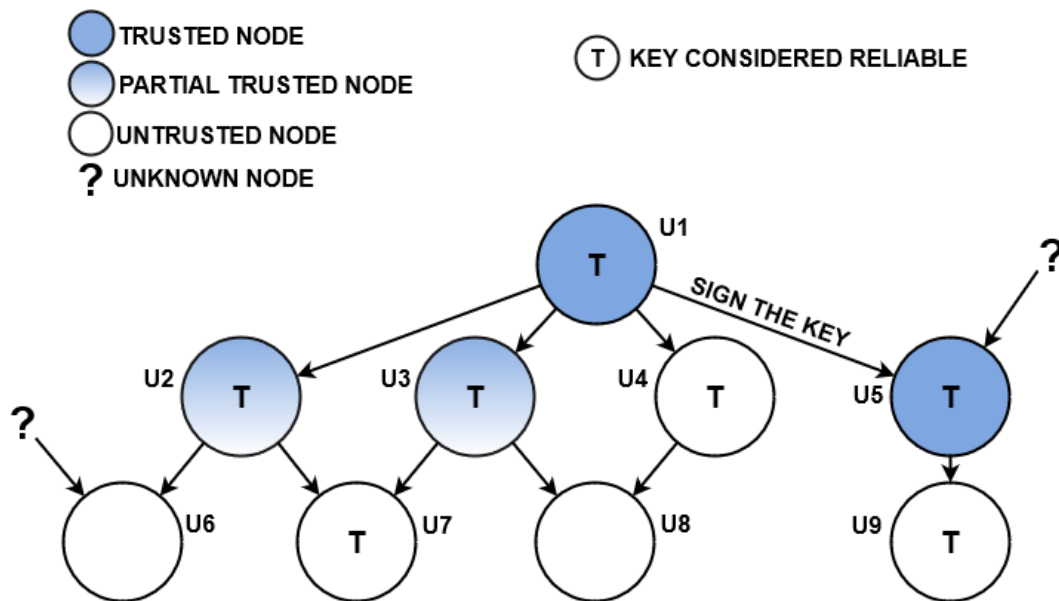
$$\begin{aligned}
 K &= (u_B)^{r_A} \bmod p \\
 &= (g^{r_B} \bmod p)^{r_A} \bmod p \\
 &= (g^{r_B})^{r_A} \bmod p \\
 &= g^{r_B r_A} \bmod p \\
 &= (g^{r_A})^{r_B} \bmod p \\
 &= (g^{r_A} \bmod p)^{r_B} \bmod p \\
 &= (u_A)^{r_B} \bmod p \quad \square
 \end{aligned} \tag{2.1}$$

## 2.5 Pretty Good Privacy (PGP)

PGP [ZIMMERMANN, 1995] provides encryption and digital signature mechanisms through public key cryptography. However, an important feature of this system is that the authentication of users' keys is not centralized in a fully trusted Certificate Authority, like in a Public Key Infrastructure (PKI) model. Instead, it is done in a distributed way, through a “web of trust” from which the network users themselves verify the authenticity of each other's public keys and then sign the certificates they trust. Besides that, each user can assign a degree of trust to their peers, which indirectly defines the degree of trust in the public keys contained in the certificates signed by these trusted users.

An example of a web of trust is illustrated in Figure 7.

Figure 7: PGP web of trust



Source: Authors.

In this figure,  $U_1$  checks the public keys of users  $U_2$  to  $U_5$  and then signs the corresponding certificates. Additionally, it assigns a level of trust to each of these users. For example, when assigning full trust to the user  $U_5$ , all certificates signed by this user (e.g.,  $U_9$ ) are automatically trusted. On the other hand, when the level of trust is partial, two or more signatures are required for the certificate to be considered valid (e.g., the signatures of  $U_2$  and  $U_3$  on the  $U_7$ 's certificate make it be considered valid if the trust in  $U_2$  and  $U_3$  is 50%). Trust levels are assigned by each user, similar to trust relationships on social networks.

## 2.6 Distributed Hash Table (DHT)

Similar to a common hash table, a distributed hash table (DHT) is a data structure for efficient storage and search of information organized in the form  $(K, V)$ , where  $K$  is the indexing key and  $V$  is the corresponding value [ZHANG et al., 2013]. A distinguishing feature of DHTs is that their content is distributed among the various nodes that make up the network instead of being on a single node. More precisely, nodes whose identifiers are closest to the indexing keys to be stored are responsible for storing the corresponding data. This proximity depends on the metric adopted by the algorithm. For greater availability, storage is usually done with some degree of redundancy (i.e., with data replication); thus, even if a node leaves the network, DHT information remains available on the remaining nodes.

Several algorithms can be used to build a DHT network, such as Kademlia [MAY-MOUNKOV; MAZIÈRES, 2002] and Chord [STOICA et al., 2003]. The main difference between the algorithms is how the pairs  $(K, V)$  are indexed and which is the search strategy for information on the network.

## 3 SPECIFICATION OF PROJECT REQUIREMENTS

Instant messaging apps have many features beyond just texting. In this chapter, we will describe the relevant points for this research, define the scope of the proposed work, and specify the requirements to reach the established goals.

### 3.1 Scope

Based on the description detailed in Chapter 1, the scope of the project will include:

- Develop a communication architecture with security and integrity mechanisms that can be incorporated into IM app systems;
- Detail how to allow the IM app's users to exchange messages using a distributed P2P communication through DHT;
- Detail how to construct a web of trust using PGP in the IM app system;
- Specify the hash chain content construction detailing how to audit a conversation chat.

Furthermore, we can add to the project the development of an external system to audit a chat, checking the hash chain.

The following points present in popular market apps will be excluded from the scope:

- Sending messages containing attached files, photos, videos: This can be abstract as a text message, then the process to construct the hash chain will be the same;
- Group chats: For this work, we will develop the features for a pair of interlocutors. However, because of the facility and independency to construct the hash chain, it is possible to expand our proposal to a group chat, similar to Bitcoin cryptocurrency blockchain records;

- Cloud backup system: If the user wants to save a backup of the chats in a cloud system, it is possible to upload the files from the device. However, our project will not do this backup by default. The messages' content privacy will be ensured because of cryptography. This content just will be exposed when an interlocutor reveals the messages;
- Web and desktop version: Some IM apps, for example, WhatsApp and Telegram, present web and desktop versions. For this project, we will not include these features. We will focus our architecture on the mobile app version only.

## 3.2 Requirements

Project requirements are the conditions, features, and tasks that need to be completed to ensure the success or achievement of the project. Aiming at the proposed goals and the scope detailed above, we define the following functional and non-functional requirements for this project.

### 3.2.1 Functional Requirements

Functional requirements are those that describe the actions that the system must perform. For this project, the following functional requirements were listed:

- The user must register in the system on its first startup;
- The system must connect the new user to the communication network;
- The system must generate the new user's keys and announce the public keys to the network;
- The system must change a user from "OFFLINE" to "ONLINE" status when the user connects to the communication network;
- User can add a new contact;
- User can chat with a contact;
- In the conversation, the user can send text messages;
- In the conversation, the system must generate the message's block according to the hash chain's rules;

- When a user  $\mathcal{A}$  sends a message to a user  $\mathcal{B}$  ONLINE, user  $\mathcal{B}$  should receive the message that should be displayed in the conversation;
- When a user  $\mathcal{A}$  and a user  $\mathcal{B}$  send a message simultaneously in the same chat, the system must handle the conflict in the hash chain construction;
- User can verify the digital certificate of other contacts;
- User can sign the digital certificate of contacts that he trusts;
- The user can verify the integrity of their conversations and prove it to a third party.

### 3.2.2 Non-Functional Requirements

Non-functional requirements are features and constraints that must be present in the system, but can not always be achieved by just building code functions. Non-functional requirements depend on how the system was designed and its physical structure. For this project, the following non-functional requirements were determined:

- The system must guarantee the integrity of the messages exchanged;
- The system must guarantee the integrity of the conversation records;
- The system must guarantee the confidentiality of messages exchanged on the network;
- The system must guarantee the privacy of the content of the messages until one of the conversation party expose this content to audit;
- The system must be resilient and resistant to availability attacks;
- The system must present good usability [NAYEBI; DESHARNAIS; ABRAN, 2012] with an acceptable time performance for communication, in the order of units of seconds.
- The system must be scalable to support millions of users, similar to the apps available on the market.



### 3.3 Application scenario and target audience

Due to the motivation of this work and the defined requirements, the principal target audience of this work will be the users of IM apps that use this system for business. As business, we consider any conversation that involves agreements between the parties, whether that agreement is financial or not. We will focus on an application scenario when the parties need a conversation record as proof, so the agreement is not just lip service based on good faith.

Some application scenarios can be:

- conversation of buying and selling assets;
- official information exchange;
- conversation between customer and service provider that requires protocol record.

Beyond these scenarios, if an IM app user just wants to securely record his conversation with more integrity and verification mechanisms, this user can use the solution proposed in this work.

## 4 RELATED WORKS

To guarantee the integrity and authenticity of the messages exchanged, most security protocols currently used for instant communication do not involve digital signatures, but only schemes based on symmetric keys [WHATSAPP, 2016; COHN-GORDON et al., 2017], like Messages Authentication Codes (MACs) and Authenticated Encryption (AE) [SIMPLICIO et al., 2013]. This is the case with popular apps like WhatsApp [WHATSAPP, 2016], Signal [COHN-GORDON et al., 2017], Telegram [TELEGRAM, 2019], and Threema [THREEMA, 2021]. For this reason, access to received messages also allows their manipulation. To manipulate local content, it would be enough:

1. access the memory space where the messages are stored,
2. manipulate the message, and
3. recalculate the authentication data set of the modified message, using the same key used to verify the original content.

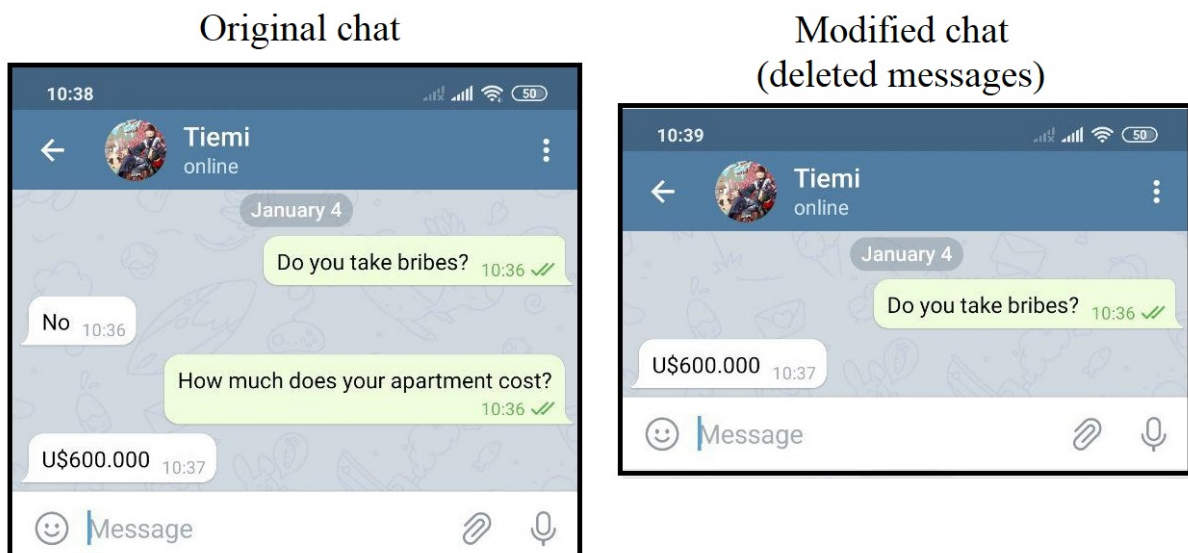
In this case, although it is possible to identify discrepancies between the contents stored on the devices of the users participating in the communication, it would not be possible to assess who was responsible for the manipulation (or even if there was a change by more than one user). This change can be made on purpose or even accidentally, due to latency problems in communication [SCHLIEP; KARINIEMI; HOPPER, 2017; SCHLIEP; HOPPER, 2018].

In cases where the encryption mechanisms used in communications adopt the client-server model, in principle, one can verify the occurrence of local manipulations when comparing them with the data stored on the server itself. However, this is not always possible in practice, depending on how the application is designed. One example, in particular, is the case of Telegram, which uses client-server encryption in conversations by default (i.e., when the "secret chat" feature is not enabled). Although the server actually stores messages exchanged between users, the application still allows data manipulation by:

1. include among its features the ability to delete messages from all users' devices and the server, whether those messages are sent by the user who is deleting or not [TELEGRAM, 2018], and
2. use an authenticated encryption mechanism to protect messages, allowing local manipulations to be imperceptible after deleting the corresponding messages on the server [TELEGRAM, 2019].

Figure 8 shows an example of how a conversation could be manipulated using simply message deletion function and getting a meaning quite different from the original.

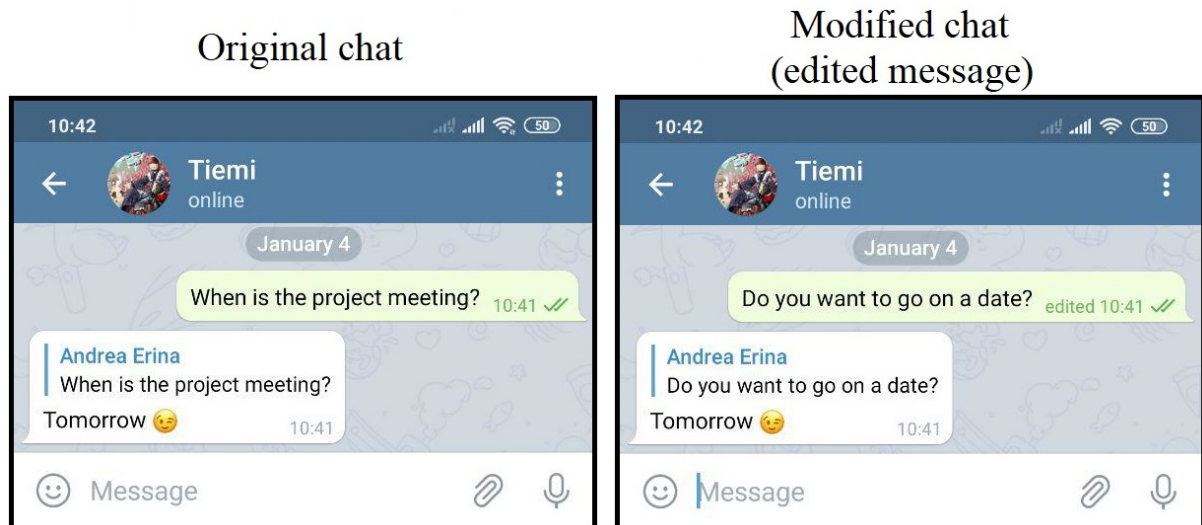
Figure 8: Example of conversation with deleted message



Source: Authors using *Telegram* app.

More complex examples would involve editing messages instead of deleting them. Figure 9 shows an example of how a conversation's meaning could be changed just by editing message's content.

Figure 9: Example of conversation with edited message



Source: Authors using *Telegram* app.

In the case of applications that use end-to-end encryption, modified conversations are even more difficult to detect. In particular, one of the main security mechanisms currently used for this purpose is the protocol developed by the open-source application Signal, which was formally analyzed in [COHN-GORDON et al., 2017] and is currently also adopted by other applications, such as WhatsApp [WHATSAPP, 2016]. The main security feature of this protocol is the high degree of confidentiality of the messages exchanged: in addition to using end-to-end encryption, the message protection keys are continually renewed through the mechanism known as “double ratchet” [MARLINSPIKE; PERRIN, 2016]. As a result of this mechanism, each message is protected by a distinct and specific key for each pair of users (or, in the case of group conversations, between each user and the group itself) [MARLINSPIKE; PERRIN, 2016; RÖSLER; MAINKA; SCHWENK, 2018]. Thus, discovering the key of a message does not allow discovering keys of past messages, a property known as “forward secrecy” or “future secrecy”; in addition, the system can “self-regenerate”, preventing future communications from being compromised even if a key is exposed.

Although this type of mechanism is important for the preservation of users’ privacy, it compromises the requirements of a corporate scenario where it is wanted to have conversations recorded as reliable documents for later consultations. After all, given end-to-end encryption, the system does not allow central servers to access the contents of messages, so there is no tamper-proof environment from which the original messages can be retrieved.

In reality, some instant messaging applications with end-to-end security even avoid having a central server, using peer-to-peer (P2P) technologies so that there are no records of communication metadata (e.g., instants of message sending) [KOMO et al., 2018; ROGERS et al., 2018; GULTSCH, 2014]. This is the case of Briar app [ROGERS et al., 2018] that focuses on independence and robustness. They defend server independence by allowing connection via Bluetooth or Wi-Fi when the Internet is down, and via the Tor network. Another reason they use the Tor network is to maintain users’ privacy and avoid possible censorship. Moreover, the messages exchanged in the Briar app are stored securely just on the user’s device by default, not in the cloud. Conversations app [GULTSCH, 2014] also focuses on a distributed network system based on XMPP federated protocol [SAINT-ANDRE, 2011] that allows the user to freely choose a trustworthy server by themselves while still chatting with contacts that are using other servers. They use TLS encryption [RESCORLA, 2018] for confidentiality between the user and the chosen XMPP server and give the user the choice to enable OMEMO [STRAUB et al., 2015] or OpenPGP [CALLAS et al., 2007] end-to-end encryption mechanisms.

To the best of our knowledge, one of the few works in the literature that takes message integrity requirements into account after receiving them in instant messaging applications is the protocol proposed in [SCHLIEP; HOPPER, 2018]. The integrity of the conversation is guaranteed by the combination of the NAXOS key agreement protocol [LAMACCHIA; LAUTER; MITYAGIN, 2007] and symmetric encryption protocol authenticated with associated data, the AES-GCM with random initialization vectors [MCGREW; VIEGA, 2004]. An ephemeral NAXOS key is generated for each message based on the previous message and then that key is used to encrypt the current message. However, the solution also seeks to provide plausible deniability, i.e., by design it does not provide the non-repudiation property, leaving an unsatisfactory gap for its use in corporate settings.

The following tables summarize and compare some of messaging apps on the market mentioned in this section. Table 2 highlights some structural points of the applications. The first point analyzed is whether the application is open or closed source. From a security perspective, open sources are better than closed, because it is possible to inspect them and prevent vulnerabilities and backdoors. The other point highlighted in this table is the connection architecture adopted. As described in Section 1.1, the system may have particular vulnerabilities depending on this architecture.

Table 2: Comparing IM apps structure

<b>App</b>	<b>System connection</b>	<b>Source code</b>
<i>WhatsApp</i>	centralized	closed
<i>Telegram</i>	centralized	open
<i>Threema</i>	centralized	open
<i>Signal</i>	centralized	open
<i>Briar</i>	distributed	open
<i>Conversations</i>	distributed	open
[SCHLIEP; HOPPER, 2018]	distributed	–

Source: Authors.

Table 3 focuses on the confidentiality and privacy aspects of conversations. A fact considered fundamental in this regard is end-to-end (E2E) encryption between the interlocutors of the conversation. Among the applications listed, we analyze which ones have this E2E encryption and which cryptographic algorithms are used.

Table 3: Comparing IM apps confidentiality and privacy mechanisms

<b>App</b>	<b>End-to-end encryption</b>	<b>Algorithms</b>
<i>WhatsApp</i>	“Yes” (questionable implementation)	Signal protocol encryption (AES256 in CBC mode, HMAC-SHA256, ECDH)
<i>Telegram</i>	Only in “secret chat”	MTPProto’s End-to-End encryption (SHA-256, AES256 in IGE mode, DH)
<i>Threema</i>	Yes	Cryptography library NaCl [BERNSTEIN, 2009] (ECDH, XSalsa20, Poly1305-AES)
<i>Signal</i>	Yes	X3DH, HMAC-SHA256, AES256 in CBC mode
<i>Briar</i>	Yes	key agreement protocols (BQP and BHP), BLAKE2b, TLS
<i>Conversations</i>	No by default	OMEMO, OpenPGP
[SCHLIEP; HOPPER, 2018]	Yes	AES-GCM with random IV, DH

Source: Authors.

As discussed extensively in this work, Table 4 raises the issue of conversation integrity

and auditability. In this Table, we analyze if there is a mechanism to verify the integrity of a conversation, how the users are identified and authenticated at the app network, and if the system/app has the concern to permit an audit in a conversation.

Table 4: Comparing IM apps audit mechanisms

<b>App</b>	<b>Conversation integrity</b>	<b>Authenticity check</b>	<b>Audit</b>
<i>WhatsApp</i>	–	Key fingerprint verification	Plausible deniability
<i>Telegram</i>	–	Key fingerprint verification	Plausible deniability
<i>Threema</i>	–	Threema ID	Anonymity, Repudiability
<i>Signal</i>	--	Key fingerprint verification	Plausible deniability
<i>Briar</i>	–	Key fingerprint verification	Plausible deniability
<i>Conversations</i>	–	Key fingerprint verification	Plausible deniability
<i>[SCHLIEP; HOPPER, 2018]</i>	NAXOS	–	Deniability

Source: Authors.

In addition to the highlighted features, Table 5 lists some more security features present in the applications.

Table 5: IM apps others security features

<b>App</b>	<b>Others security features</b>
<i>WhatsApp</i>	Forward secrecy
<i>Telegram</i>	Perfect Forward Secrecy (PFS)
<i>Threema</i>	Forward Secrecy
<i>Signal</i>	Double Ratchet
<i>Briar</i>	Tor unlinkability
<i>Conversations</i>	–
<i>[SCHLIEP; HOPPER, 2018]</i>	Forward and Backward Secrecy

Source: Authors.

Based on the goals aimed in Section 1.2, it is noted that the Briar, Conversations, and [SCHLIEP; HOPPER, 2018] solutions are close to what we want. However, none of these solutions have all the features we need to comply with the requirements raised in Chapter 3.



## 5 METHODS AND MATERIALS

Based on this work's proposal, the main materials needed are the following items. These materials will be used in our proof of concept (PoC) described in Chapter 7.

- A personal computer with access to wireless networks;
- At least three smartphones with Android operating system version 6.0 or higher;
- Wireless network access, for example, using a router with protocol IEEE 802.11g (Wi-Fi).

Based on this work's goals, the following use cases were described. They will be important to model our architecture proposed in Chapter 6.

### 5.1 Use cases

A use case [GOMAA, 2011] is a software engineering model used to facilitate the understanding and specification of a system. A use case describes how the user will interact with the system and how the system will behave. Each use case is represented as a sequence of simple steps, beginning with a user's goal and ending when that goal is fulfilled.

In the following subsections, we describe the main use cases that we want to comply with in this work. These use cases must agree with the scope and requirements described in Chapter 3.

#### 5.1.1 Use case: Register new user

This use case will describe the initial registration of a new user in the proposed IM app system.

**Actors:** New user.

**Stakeholders:** New user, system users.

**Primary actor:** New user.

**Precondition:** New user is not registered in the IM app system.

**Poscondition:** New user's connection information is available on the system DHT network for any other user consult.

**Trigger:** New user starts the IM app for the first time and the system requests a new register.

**Main success scenario:**

1. New user starts the IM app for the first time;
2. The IM app requests a username (user ID) and password for the new user;
3. New user informs a user name and password;
4. The IM app generates the new user's DSA, PGP, and DH key pairs;
5. The IM app saves safely the new user's connection information and public keys at the DHT network.

**Alternative paths:**

- If the user name informed is already used (item 3) then the IM app informs that this user name is not available and requests another user name to continue. The user name is used as the user's identifier which is why its have to be unique in the network.
- If key pairs can not be generated (item 4) then the IM app presents an error and cancels the new register.
- If the IM app can not connect with the DHT network, consequently, it can not save the new user's connection information in DHT (item 5), then the IM app informs connection problem and waits for connection to be established.

### 5.1.2 Use case: Send a message

This use case will describe what needs to happen to a user sends a message to another user in the IM app system.

**Actors:** Sender, receiver.

**Stakeholders:** Sender, receiver.

**Primary actor:** Sender

**Precondition:** The sender is connected to the IM app DHT network, the receiver is registered in the IM app's system and connected to the DHT network, and the sender knows the receiver's user ID.

**Poscondition:** The receiver can read the message that the sender sent.

**Trigger:** The sender chooses the receiver.

**Main success scenario:**

1. The sender searches for the receiver on the IM app system's DHT network.
2. The sender selects the receiver;
3. The sender's app opens the chat screen;
4. The sender writes the message;
5. The sender hits the send button;
6. The sender's app prepares the message block;
7. The sender's app sends the message block;
8. The receiver's app receives the message block;
9. The receiver's app decodes and verifies the message block;
10. The receiver's app shows the message;
11. The sender's app shows the message;
12. The receiver reads the message.

**Alternative paths:**

- If the message block can not be generated (item 6) then the system catches an error and retry to generate the message block.
- If the message block can not be decoded (item 9) then the system catches an error and informs the sender to resend the message.

- If the message block check shows as invalid (item 9) then the system discards this message block and does not shows the message.

### 5.1.3 Use case: Trust another user

This use case will describe how a user can trust another user and cooperate to create the PGP web of trust.

**Actors:** User  $\mathcal{A}$  who will trust User  $\mathcal{B}$ .

**Stakeholders:** All users in the IM app.

**Primary actor:** User  $\mathcal{A}$ .

**Precondition:** User  $\mathcal{A}$  does not trust User  $\mathcal{B}$  in the app system.

**Poscondition:** User  $\mathcal{A}$  trusts User  $\mathcal{B}$ .

**Trigger:** User  $\mathcal{A}$  verifies who trusts User  $\mathcal{B}$ .

**Main success scenario:**

1. User  $\mathcal{A}$  searches for User  $\mathcal{B}$ ;
2. User  $\mathcal{A}$  requests to verify who trusts user  $\mathcal{B}$ ;
3. User  $\mathcal{A}$ 's app shows the User  $\mathcal{B}$ 's public PGP key and the list of users that trust User  $\mathcal{B}$ ;
4. User  $\mathcal{A}$  compares if the User  $\mathcal{B}$ 's public key is trustworthy or User  $\mathcal{A}$  believes in the trust of other users listed;
5. User  $\mathcal{A}$  confirms that he trusts User  $\mathcal{B}$ ;
6. User  $\mathcal{A}$ 's app updates the system network;
7. User  $\mathcal{A}$  is displayed in the list of users who trust User  $\mathcal{B}$ .

**Alternative paths:**

- If the public key shown and the public key known are different (item 4) then User  $\mathcal{A}$  does not trust User  $\mathcal{B}$ .
- If there are no users listed that trust User  $\mathcal{B}$  (items 3 and 4) then User  $\mathcal{A}$  must verify User  $\mathcal{B}$ 's public key before confirming it's true.

### 5.1.4 Use case: Audit a chat

**Actors:** Interlocutor  $\mathcal{A}$  and the auditor.

**Stakeholders:** Interlocutor  $\mathcal{A}$ , interlocutor  $\mathcal{B}$ , the auditor, and anyone outside the conversation that wants to confirm what was chatted about.

**Primary actor:** The interlocutor that wants to show the chat for the auditor, in this case, let's suppose that is interlocutor  $\mathcal{A}$ .

**Precondition:** The auditor does not know the messages exchanged between interlocutor  $\mathcal{A}$  and interlocutor  $\mathcal{B}$ 's chat.

**Poscondition:** Anyone outside the conversation (represented by the auditor) comes to confirm the content of the conversation.

**Trigger:** Interlocutor  $\mathcal{A}$  wants to prove to the auditor what was chatted between interlocutor  $\mathcal{A}$  and interlocutor  $\mathcal{B}$ .

**Main success scenario:**

1. Interlocutor  $\mathcal{A}$  shows to the auditor the messages that  $\mathcal{A}$  wants to be audited, informing messages' plain text, related hashes, and signatures.
2. The auditor reconstructs the messages' blocks (hash chain sequence);
3. The auditor compares if the generated hashes are equal to the received hashes;
4. The auditor verifies if the signatures are genuine;
5. The integrity, authenticity, and non-repudiation of the conversation between interlocutors  $\mathcal{A}$  and  $\mathcal{B}$  are proven.

**Alternative paths:**

- If the reconstructed and the received hashes chain are different (item 3) then the reported conversation does not match the real conversation.
- If the signature is fake (item 4) then the information received from interlocutor  $\mathcal{A}$  can not be used to audit.

## 6 PROPOSED ARCHITECTURE

The proposed architecture has two main parts: the connection network between users and the underlying cryptographic security structure. The connection network, based on DHT, is responsible for managing the information that allows the connection between users. The security structure guarantees the confidentiality, integrity, and authenticity of messages, as well as the identification of interlocutors and the auditing of conversations.

As it is not a new communication protocol, the proposed architecture can be incorporated in parts by others IM app systems. The modules of distributed connection with DHT, PGP web of trust, and chat integrity with hash chain are all independent and each one has a specific contribution to meet the requirements.

### 6.1 Connection network

One of the roles of the central server in existing instant messaging applications is to facilitate the connection between users. In this work, the connection will be done through a DHT network in a distributed architecture. This network will store the users' connection information and, thus, make P2P communication viable.

More precisely, each user must select a unique username `uname` on the network. For each user, two pairs of index keys ( $K$ ) and value ( $V$ ) will then be stored:

1.  $K = \text{Hash}(\text{uname})$ : your public domain key  $K_D$  is stored in  $V$ . This allows the creation of “protected domains”, in which only the owner of  $K_D$  can edit the information stored in the node indexed by  $K_D$  (e.g., for which  $K = \text{Hash}(K_D)$ ). This also ensures that only one user can be associated with each `uname`: if the name already exists, the network refuses to change the corresponding public key  $K_D$  unless that change request is signed by  $K_D$  itself, similarly to what occurs in the Self-certifying File System (SFS) [MAZIÈRES, 2000].
2.  $K = \text{Hash}(K_D)$ : the necessary information is stored in  $V$  to establish communication

with the user and verify his identity, also in the form of a protected domain using  $K_D$ . Specifically, the prototype will store the user’s IP address and port, as well the user DH and PGP public key and corresponding certificates to protect communications between users. This gives users support for mobility since the owner of  $K_D$  (and only him) can change their connection information at any time.

The following Table 6 summarize the **key-value** pair of information stored in the DHT for each user in the app system. Item 2 described above was divided into lines from 2 to 4 of the following table, and each key was differentiated using a keyword (id). This division was made only by choice of project and organization, these three information can be stored in a single “object” with three attributes using only the  $K_D$  key as described in item 2.

Table 6: Information stored in the DHT

Nº	Key <sup>3</sup>	Value
1	uname	User’s ECDSA public key <sup>△</sup>
2	idAddress + User’s ECDSA public key	User’s IP address and connection port <sup>△★</sup>
3	idPGP + User’s ECDSA public key	User’s PGP public key <sup>△★</sup>
4	idECDH + User’s ECDSA public key	User’s ECDH public key <sup>△★</sup>

Source: Authors

△ Protected write, ★ Protected read.

Note that ECDSA, PGP, and ECDH algorithms need an asymmetric key pair to work. This key pair does not necessarily have to be individual to each algorithm, it is theoretically possible to use the same key pair for all three in the system. However, we propose separation for convenience reasons, since each key in principle has different purposes and, therefore, may have different configurations (e.g., expiration dates).

Using this infrastructure, when a user  $\mathcal{A}$  wants to communicate with  $\mathcal{B}$ , a DHT search is executed based on some of the existing algorithms (e.g., Chord, Kademia). When obtaining the connection information of user  $\mathcal{B}$ ,  $\mathcal{A}$  can create a direct P2P connection via socket, so that all communication goes directly between the parties without a central server to forward messages. This approach avoids bottleneck problems and protects the service from possible DoS attacks.

To communicate with any user on the app system, you only need to know at least one

---

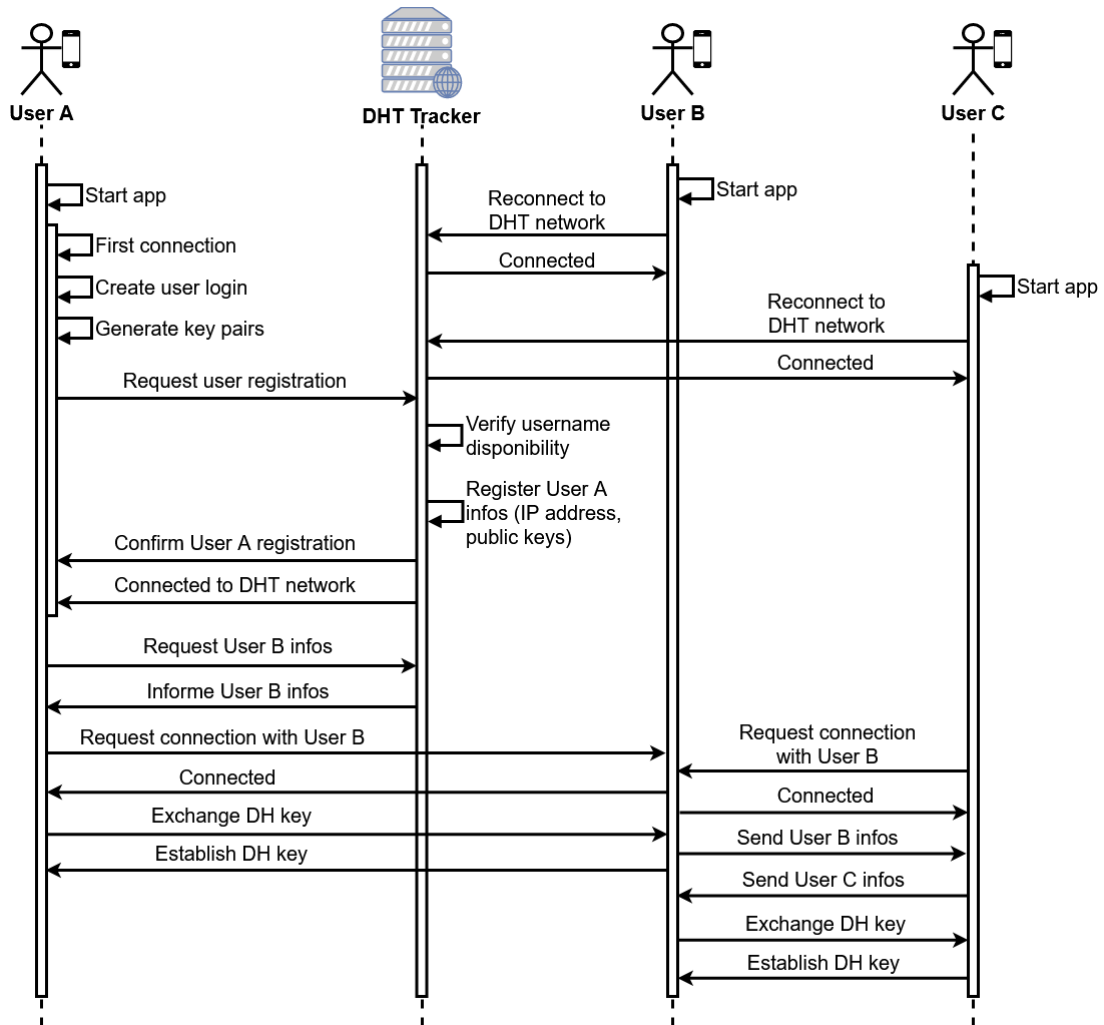
<sup>3</sup>Remember that the actual key are the hashes of the information presented in key column.

user on the DHT network. For example, this can be done through the invitation of a user who already uses the application, or through web pages, to facilitate the sign up of new users. However, this work's prototype will use a tracker server, a device commonly used in P2P networks for this purpose. Thus, the tracker is a node (or set of nodes) with a fixed IP address known initially to all nodes entering the network, as this information is directly recorded in the application itself. Although trackers create some degree of centralization in the network, their unavailability only delays the entry of new nodes (users) in the network without actually preventing such entries or affecting communications between previously registered users.

Figure 10 shows how the User  $\mathcal{A}$  establish a connection with the tracker server to obtain User  $\mathcal{B}$ 's information and then establish a connection with the User  $\mathcal{B}$ . This diagram also shows how the User  $\mathcal{C}$  can establish a direct connections with the User B without using the network tracker.



Figure 10: Sequence diagram of proposed app's connection architecture



Source: Authors.

## 6.2 Security structure

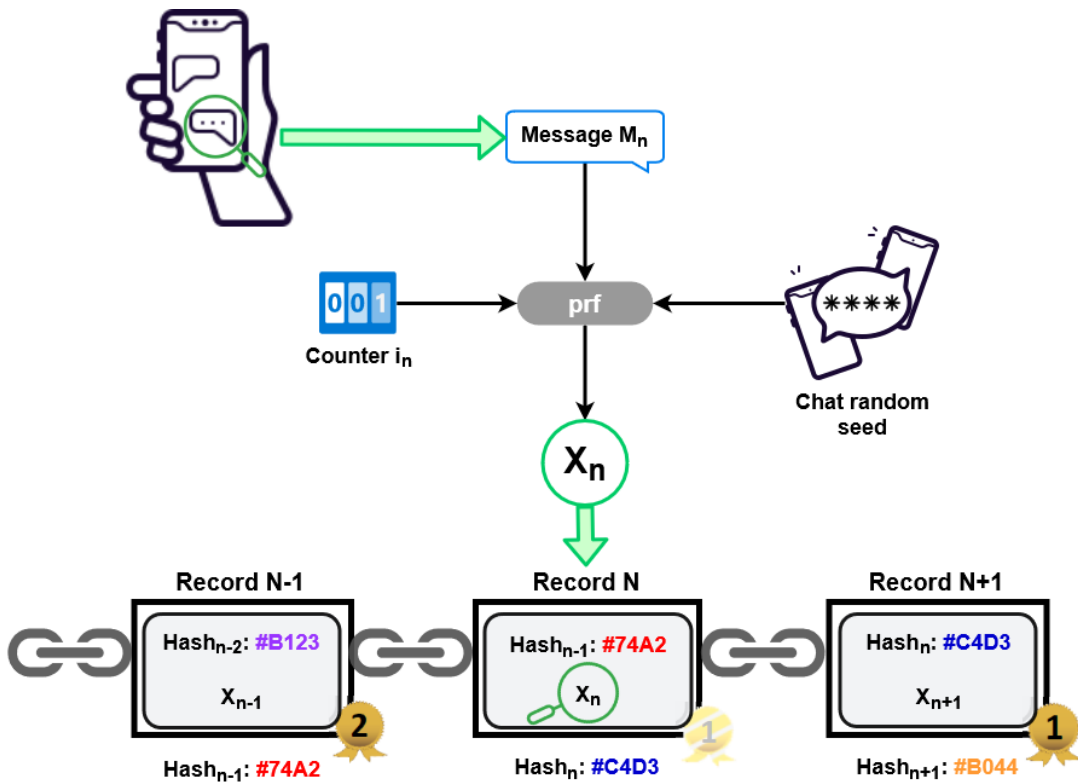
Using a DHT network, we can remove the dependency of a central server and distribute the service, improving the availability and resilience of the service, but this does not guarantee the security of the communications itself. For this, we propose to use a PGP web of trust whose information (public keys and certificates) will be stored in DHT as described in Section 6.1. Then these keys will be used to encrypt the messages, sign the message's blocks and verify the authenticity of the parties, according to the degree of trust defined by the PGP user.

In order to ensure auditability in instant messaging systems, this work proposes to use chained and signed cryptographic hashes [HU; JAKOBSSON; PERRIG, 2005]. Consid-

ring a sequence with  $\mathcal{N}$  registered messages, to insert a new message  $\mathcal{N}+1$  it is necessary that the new record presents the new message and the cryptographic hash of the previous record  $\mathcal{N}$ . The last information is then signed, allowing any manipulation to be detectable.

In addition, instead of saving the message's plain text in the blocks, we propose to use a pseudorandom function (PRF) to save messages' identifiers  $\mathcal{X}$ , for example, using a hash function and saving message's hash in the blocks. This allows a **selective disclosure** of messages, improving system **privacy**, so that only the messages to be audited are exposed and not the entire conversation. Thus, the chain grows from left to right, as illustrated in Figure 11.

Figure 11: Hash chain structure for messages. Assuming that user 1 sent messages  $\mathcal{N}$  and  $\mathcal{N}+1$ , only his last signature needs to be stored



Source: Authors.

More formally, we have the following logic for creating the chain of messages of a conversation (note that each  $block_n$  is digitally signed):

1.  $block_0 = (\emptyset, X_1)$ , where  $\emptyset$  is a string of bits filled with zeros, indicating the beginning of the communication, and  $X_1 = prf(M_1 + seed + i_1)$  when  $M_1$  is the conversation's

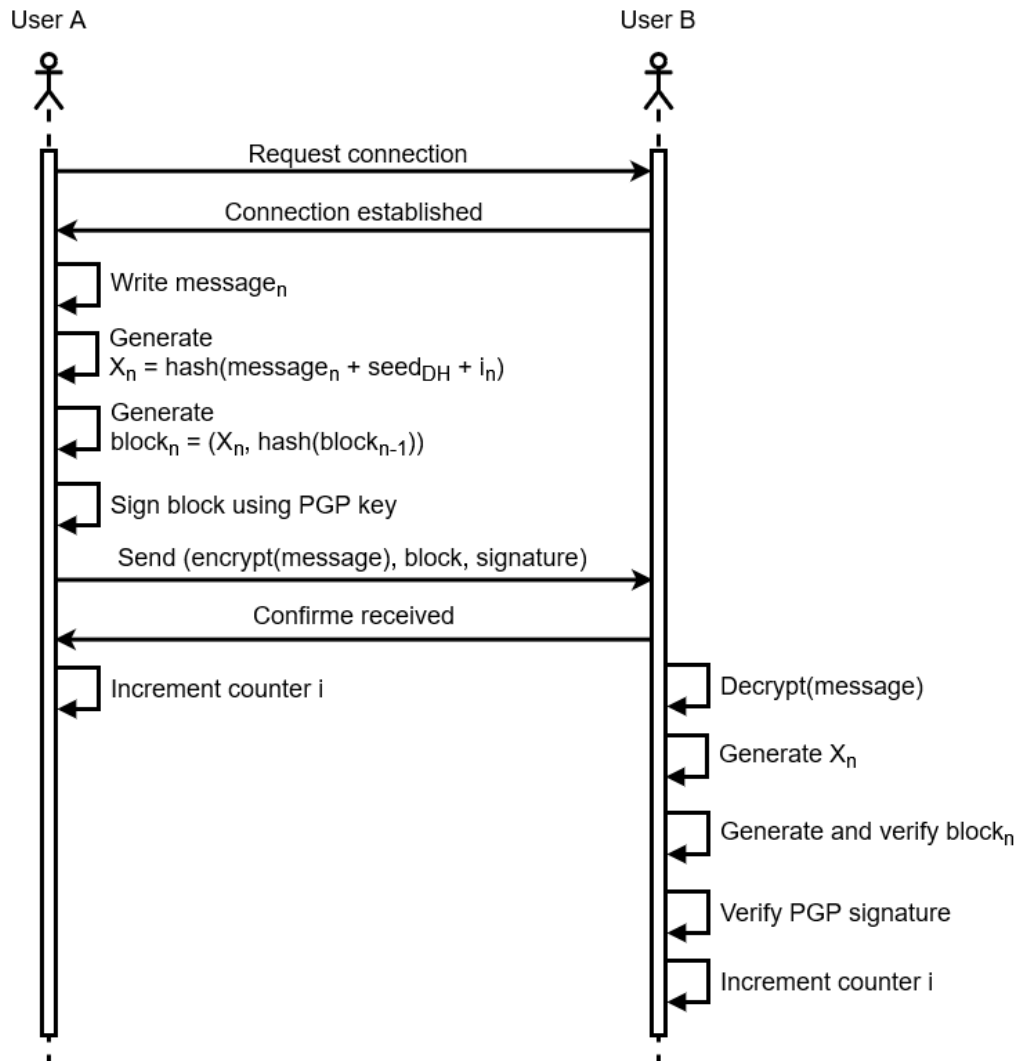
first message,  $seed$  is the DH key exchanged in the chat, and  $i$  is the counter of messages in this chat;

2.  $block_n = (h_{n-1}, X_n)$ , for  $n \geq 1$ , where  $h_{n-1} = Hash(block_{n-1})$  and  $X_n = prf(M_n + seed + i_n)$  when  $M_n$  is the conversation's  $n$ -th message.

Notice that we propose to use  $\emptyset$  zeros values over an Initialization Vector (IV) in the first block. A reason for this choice is that we do not need a security number in this part to ensure the security of the system, so the  $\emptyset$  value is enough and demands fewer computer resources than an IV. Another reason is that with the  $\emptyset$  we can be sure that it is the start of the chain, preventing improper messages to be added before the first register.

Figure 12 shows what happens when a User  $\mathcal{A}$  sends a message to a User  $\mathcal{B}$ . First, we have the processes in  $\mathcal{A}$  to generate the message. User  $\mathcal{A}$  will input the message that he wants to send. Then the system will generate the value  $\mathcal{X}$  related to this message, generates this message's block, and finally signs this block with User  $\mathcal{A}$  private PGP key. The message will be end-to-end encrypted using the User  $\mathcal{B}$  public PGP key and be sent together with the message's block and this signature. When User  $\mathcal{B}$  receives this data, the system decrypts the message with User  $\mathcal{B}$  private PGP key and generates the value  $\mathcal{X}$  and the message's block related to this message. Next, it will compare the block received with the block generated and verify the signature received using User  $\mathcal{A}$  public PGP key. If everything is correct, this message is saved and displayed to User  $\mathcal{B}$ .

Figure 12: Sequence diagram of proposed app's conversation dynamics



Source: Authors.

Due to the properties of cryptographic hashing algorithms, any change in the input data leads to changes in the algorithm's output hash. In this way, a sequential link is created that ensures the integrity of each individual record and also the order of the records. At the same time, if the last signature in the chain is valid, this is sufficient to ensure the complete conversation's non-repudiation, so that previous signatures of the same user can be discarded, optimizing memory use. Signatures of other users, who sent previous messages, must be preserved to ensure non-repudiation up to that point of the communication. By using such structure for exchanging messages between users, it allows reliable audits of conversations by anyone who has access to a sequence of exchanged messages, just reconstructing the chain and checking each block's hashes and final signatures.

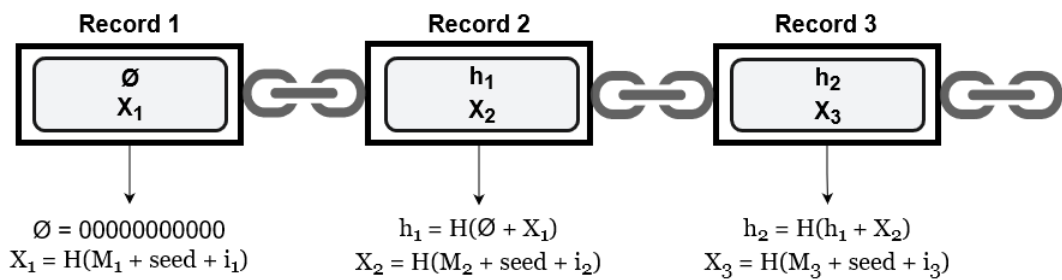
If the intention is to reveal only a portion of messages, omitting previous and sub-

sequent blocks, then the integrity check of the revealed content is still possible as long as the digital signature for the last block is available. In this case, only the value  $\mathcal{X}$  of the omitted messages would be available for analysis, which would not allow the recovery of the message itself except for possible brute force attacks (e.g., testing several possible messages).

### 6.2.1 Analysis of the records' integrity with the proposed hash chain

Suppose a small part of the conversation, as illustrated in Figure 13. In this scenario of instant messaging, the following attacks are prevented by the proposed structure, assuming that at least one of the interlocutors is honest.

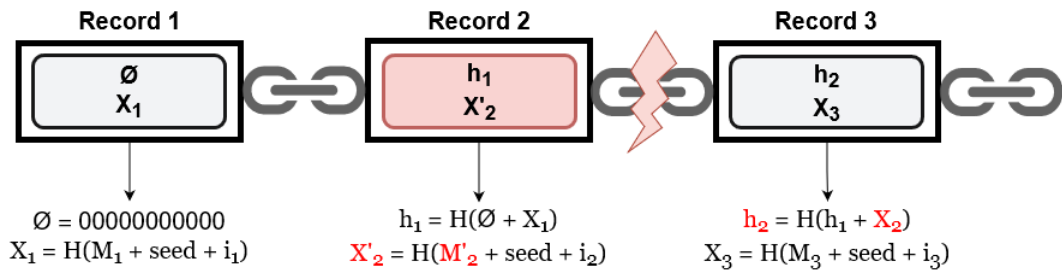
Figure 13: Messages chat hash chain



Source: Authors.

- *Editing message content*: Replacing the message  $M_n$  with a message  $M'_n$ , where  $M_n \neq M'_n$ , we have  $h_n = H(h_{n-1} + X_n)$  and  $h'_n = H(h_{n-1} + X'_n)$ , so that  $h_n \neq h'_n$ . When calculating the hash of the record  $n$  with the edited message  $M'_n$ , you get  $h'_n$ , which is different from the value  $h_n$  saved in the record  $n + 1$  and digitally signed. Figure 14 presents an example of a scenario in which  $M_2$  is changed to  $M'_2$ . When calculating the hash of record 2, you get  $h'_2$ , which differs from the value  $h_2$  saved in record 3 with overwhelming probability, it's evidenced a change between records 2 and 3.

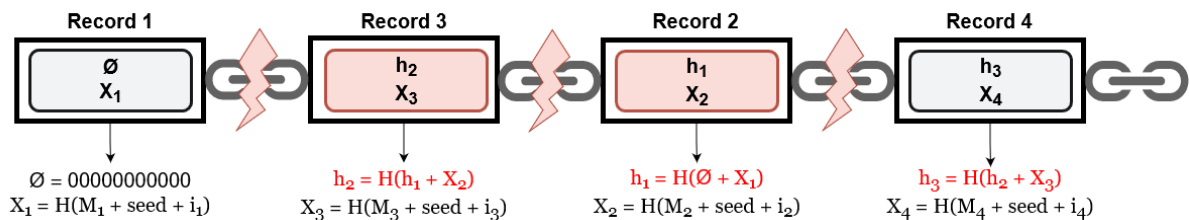
Figure 14: Conversation with an edited message



Source: Authors.

- *Changing the messages' order*: Given the sequence of messages  $M_{n-1}$ ,  $M_n$ ,  $M_{n+1}$ , and  $M_{n+2}$ , suppose the order change between the messages  $M_n$  and  $M_{n+1}$ . The change generates three points of divergence from the hashes: just before the records exchanged, between the records exchanged, and just after the records exchanged. Figure 15 shows as an example a change of  $M_2$  with  $M_3$  positions. When calculating and comparing the hashes sequence, there is an incompatibility between the records, as illustrated.

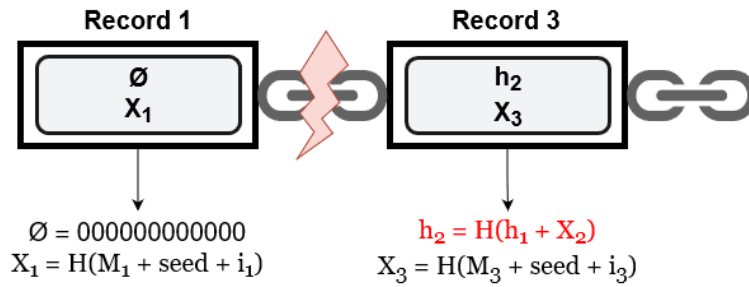
Figure 15: Conversation with messages' order changed



Source: Authors.

- *Removing messages*: If some message  $M_n$  is removed from the conversation, then the sequence of records would jump from  $M_{n-1}$  to  $M_{n+1}$ . The record  $M_{n-1}$  generates the hash  $h_{n-1}$ , while the record  $M_{n+1}$  has the hash  $h_n$  saved, thus showing the change. Figure 16 shows the removal of the message  $M_2$  as an example. When calculating the hash of record 1, you get  $h_1$ , which differs from the value  $h_2$  saved in record 3. Thus, a change between records 1 and 3 is detected.

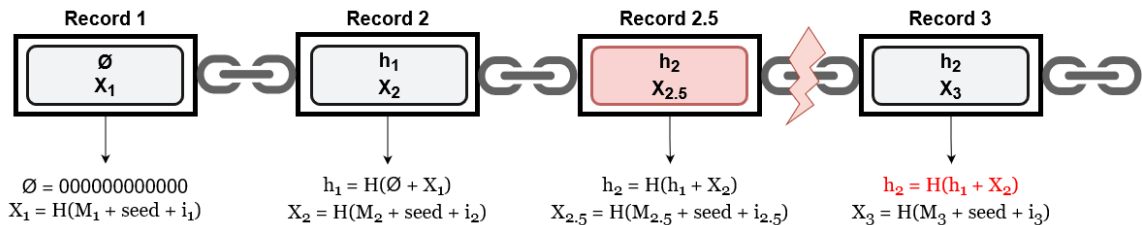
Figure 16: Conversation with deleted message



Source: Authors.

- *Inserting messages*: If the message  $M_{n.5}$  is inserted in the conversation, then the sequence of records would be  $M_n$ ,  $M_{n.5}$ , and  $M_{n+1}$ . The new record of the message  $M_{n.5}$  must contain the hash  $h_n$  to keep the sequence consistent. However, the record  $M_{n+1}$  also has the hash  $h_n$  and not  $h_{n.5}$ , showing the change. Figure 17 shows the insertion of the message  $M_{2.5}$  as an example. When comparing the hash of record 2 with the one saved in record 2.5, it is noticed that they are compatible. However, the  $h_{2.5}$  hash diverges from the  $h_2$  value saved in record 3, showing that there is some change between records 2.5 and 3.

Figure 17: Conversation with an inserted message



Source: Authors.

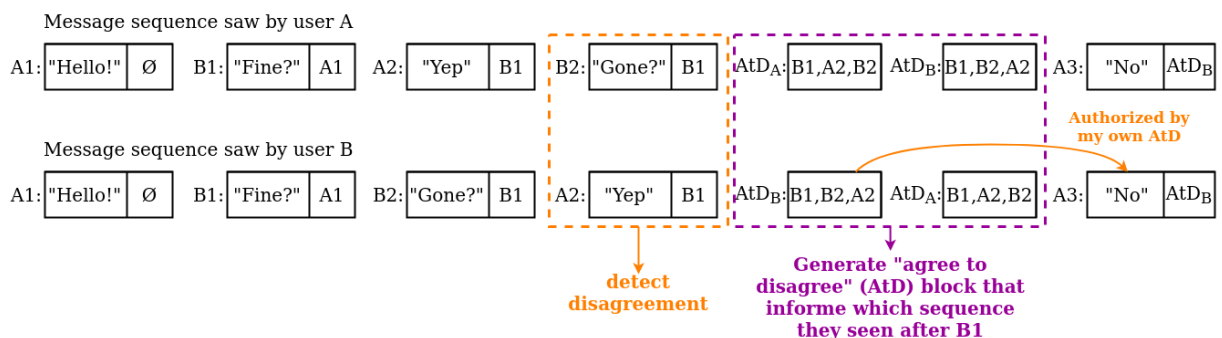
It is relevant to note that, in all examples, the architecture is robust for identifying violations of conversation integrity, although it cannot (and is not its purpose) reverse the detected changes. Besides, what guarantees that the hashes of the altered messages cannot replace the original hashes in the chain is the fact that the block at the end of the chain is always digitally signed by the sender of that message. At the same time, this guarantees authenticity and non-repudiation in communications.

## 6.2.2 Collision case: Agree-to-Disagree algorithm

In addition to all the cases raised before, there is one more case that we need to consider: messages collision. If User  $\mathcal{A}$  and User  $\mathcal{B}$  send messages simultaneously, during the constructions of these messages' blocks, both will refer to the same previous block. The way the system was specified until here, this collision would be considered a wrong block, and the received messages would be discarded, however, we know that these messages are authentic and just a collision happened. To handle these cases we propose a solution named "agree to disagree" (AtD) that will accept a punctual and temporary fork in the chain.

Figure 18 demonstrates an example of collision and how this is handled with the AtD. In the first line, we have the order of blocks seen by user  $\mathcal{A}$  and the second line is the order from user  $\mathcal{B}$ 's view. For ease of understanding, in this diagram, we constructed the blocks with the message text on the left and the previous block pointer on the right. Block written by User  $\mathcal{A}$  is identified as  $A_n$  and by User  $\mathcal{B}$  as  $B_n$ . The chain is the same for both users until block B1 then we detect a collision between blocks A2 and B2, because both points to block B1 as the previous block. Afterward, recognizing the disagreement in the chain, the system will generate an "agree to disagree" (AtD) special block that will inform which sequence they see after B1. And finally, the next block generated after the AtDs blocks will officialize the accepted sequence, remembering that each block is digitally signed. In this case, User  $\mathcal{A}$  created the block A3 that pointed to  $AtD_B$ , making official the sequence presented in the User  $\mathcal{B}$  view.

Figure 18: Example of message blocks' collision



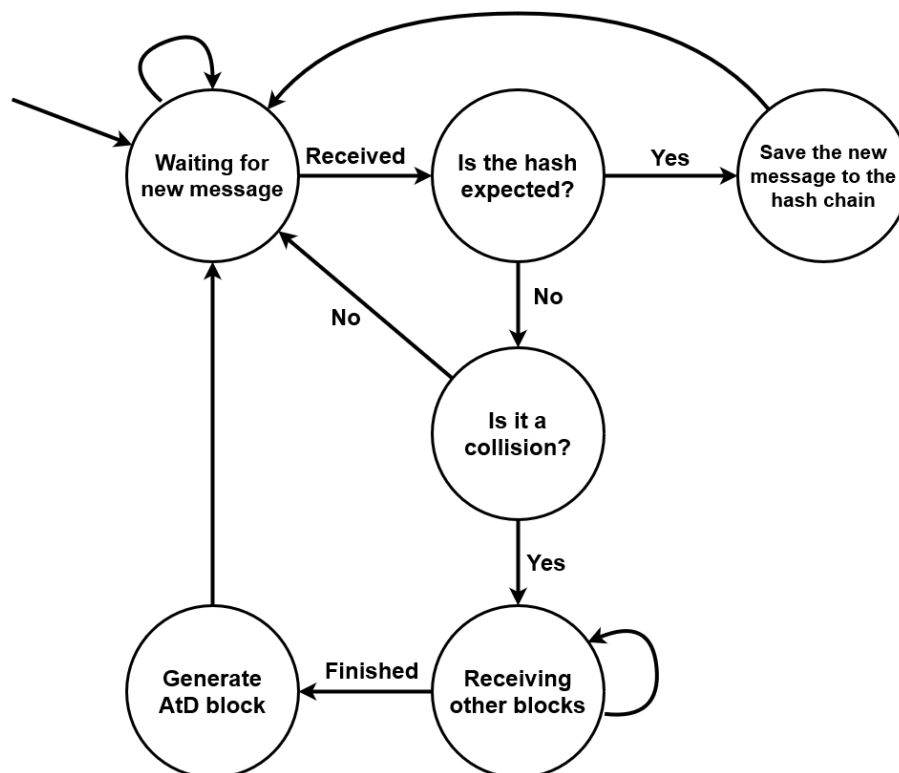
Source: Authors.

In Figure 19, we have a state machine showing how the system will deal with the blocks in the different possible situations. The system will be waiting for a new message in the initial state, and when this message is received we can have the following cases:



- Ideal case: The system confirms that message, hash, and signature are correct, then it saves all the data and returns to the initial state.
- Collision case: The system identifies a hash collision and confirms the message is authentic by the signature. Then the system will stay in a specific state to deal with the collision. In this state, it will receive and verify all blocks (can be more than one after collision detection) then at the end the system will generate and send the special block AtD and return to the initial state.
- Invalid case: If the system identifies some conflicting information in the message decryption, signature, or block's hash, it will discard this message and return to the initial state.

Figure 19: State machine for handling message blocks' collision



Source: Authors.

## 7 PROOF OF CONCEPT

Following the architecture proposed in Chapter 6, due to its complexity, we choose to develop two proof of concept (PoC) for this work. Each PoC was developed focusing on different tests described in Chapter 8.

### 7.1 Android app PoC

At first, we developed an Android app as a PoC, and the technologies and software used for this implementation were:

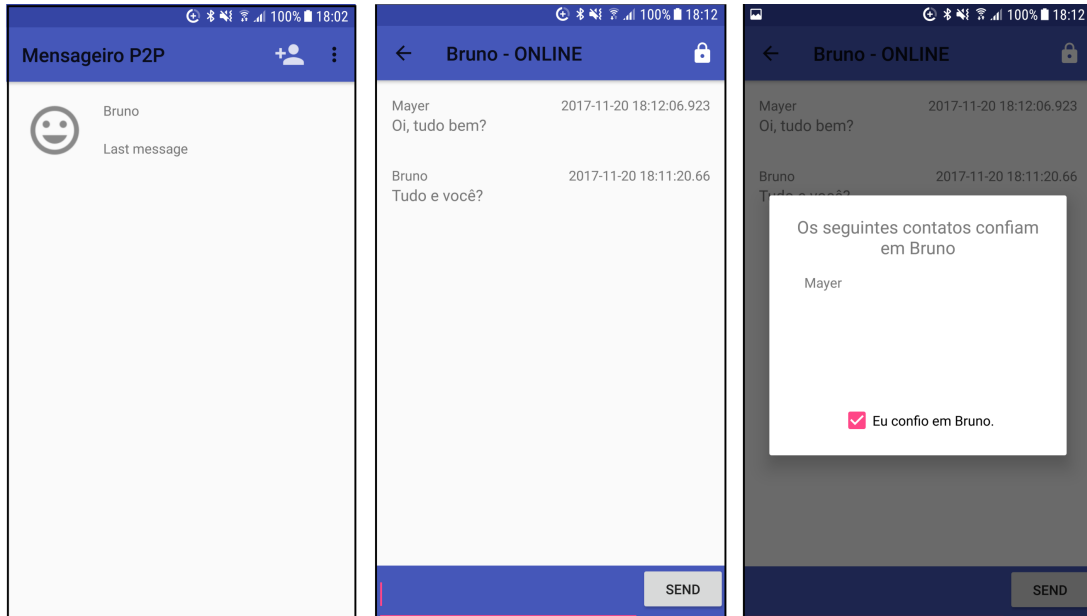
- Distributed messaging app with P2P communication developed in a course conclusion monography presented to Escola Politécnica of Universidade de São Paulo in 2017 [ARAKAKI; LEVY, 2017], available in the following GitHub repository <<https://github.com/brunoarakaki/mensageiro-p2p>>;
- Java SE Development Kit 8;
- Development environment Android Studio version 3.0.1;
- Development environment Eclipse Neon and Oxygen.

The implementation was carried out on the Android operational system, given that this platform is almost 6 times more widespread than the second largest competitor, iOS [GARTNER, 2018], in addition, Android has extensive documentation and a very active developer community.

The base app [ARAKAKI; LEVY, 2017] uses two open source libraries: TomP2P [BOCEK, 2004] and Spongy Castle [TYLEY, 2014]. Basically, the TomP2P library is responsible for creating the DHT network using the Kademlia algorithm, while the Spongy Castle library manages user certificates and digital signatures, and encrypts messages

using PGP. Because of some library functions, the application requires an Android operating system version 6.0 or higher. Figure 20 shows some screens from the application developed by [ARAKAKI; LEVY, 2017].

Figure 20: Screenshot of the app developed by [ARAKAKI; LEVY, 2017]



Source: [ARAKAKI; LEVY, 2017].

Modifying the [ARAKAKI; LEVY, 2017] app, we insert the hash chain structure to create the whole conversation records. For this purpose, the Spongy Castle library and the *java.security* package were also used and the SHA3-256 algorithm [NIST, 2015] was used to calculate the system hashes, and the ECDSA algorithm [JOHNSON; MENEZES; VANSTONE, 2001] with the *secp256* curve [Certicom Research, 2000] for digital signatures, giving the system a 128-bit security level. Thus, each message has the hash of the previous message and this information is displayed in hexadecimal as shown in Figure 21.

Figure 21: Screenshot of PoC Android



Source: Authors.

The messages of the conversations are saved only on the devices that exchanged the messages and the audit can only be carried out if any of the interlocutors decides to show the information to third parties who can reconstruct the hash chain and verify its veracity. It is not possible to audit the conversation without the permission of one of the interlocutors due to the end-to-end encryption used in communications.

This PoC does not present the selective disclosure and agree-to-disagree features. This partial result was published at [KOMO; SIMPLICIO JR., 2019]. This PoC is available in the following GitHub repository <[https://github.com/Erina-chan/app\\_eri-chain](https://github.com/Erina-chan/app_eri-chain)>.

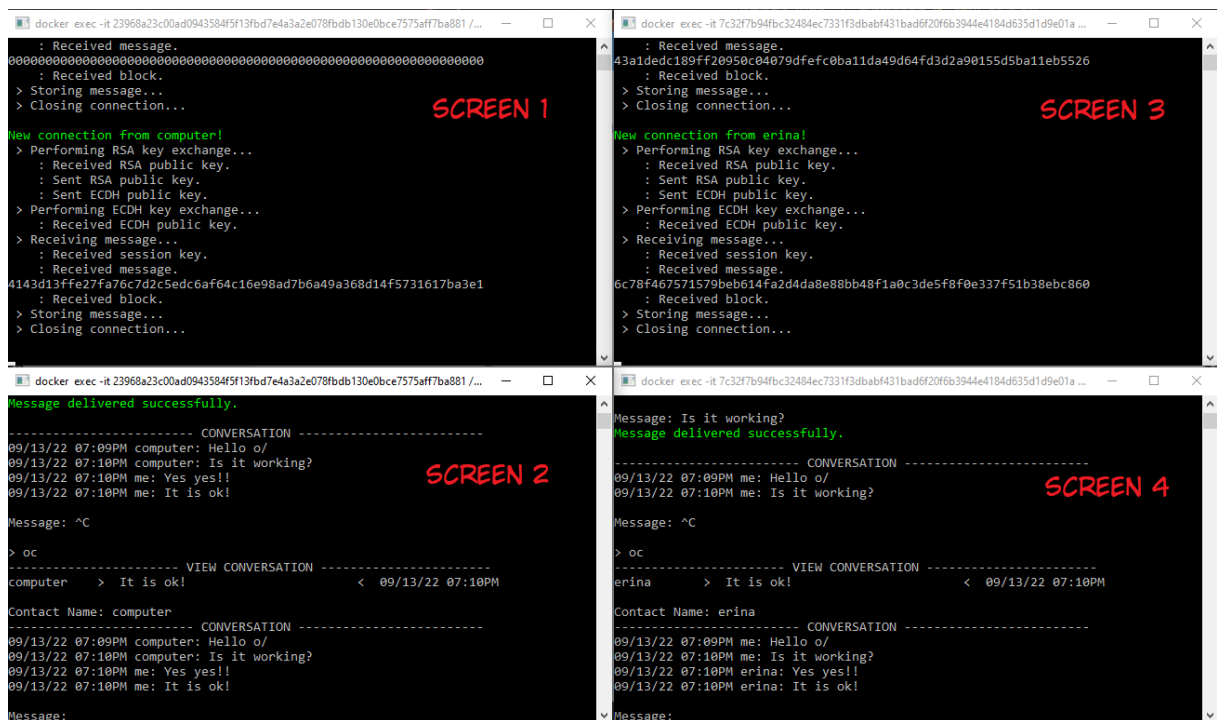
## 7.2 Python chat PoC

For hash chain audit tests we developed a PoC of an instant message chat on desktop, using Python 3 and Slyther: an encrypted peer-to-peer messaging platform written in Python, available in the following GitHub repository <<https://github.com/ajstensland/slyther>>. In this PoC the focus is the hash chain, so we inserted hash chain structure and

agree-to-disagree logic in the system, hence, it does not present the PGP web of trust. This modified version is available in the following GitHub repository <<https://github.com/Erina-chan/hashchat>>.

Figure 22 presents some screenshots from the Slyther modified, where screens 1 and 2 are user *erina*'s server and client respectively, and screens 3 and 4 are user *computer*'s server and client respectively.

Figure 22: Screenshot of PoC Python



Source: Authors.

Due to the structure of Slyther, all the records are saved in a local encrypted JSON<sup>4</sup>. We add a function that exports just the hash chain's information to another JSON file that will be accessed during the audit.

<sup>4</sup><https://www.json.org/json-en.html>

## 8 TESTS AND VALIDATIONS

As described in Chapter 7, we developed two PoC to perform different tests on each one. All efficiency tests were performed in Android PoC, the platform proposed to be used by the IM chat system. In Python PoC, we focused to prove the integrity of the chat construction and how the scheme permits to audit the chat as we proposed in this work.

### 8.1 Functional Validation Tests

After the PoC implementations, tests were carried out to verify the functioning of the systems in order to ensure that the functions initially existing were maintained and the functions developed worked as expected.

#### 8.1.1 Android PoC

Based on the [ARAKAKI; LEVY, 2017] work and the use cases raised in Section 5.1, the following tests were performed at the Android PoC:

1. User does his first registration in the app;
2. User connects on the DHT network using a tracker;
3. User establishes a connection with another user in the DHT network;
4. Users exchange messages;
5. Compare the construction of the hash chain of the conversations on users' devices;
6. Test to sign other users' digital certificate at PGP web of trust.

For this test, the tracker server was run on a notebook and the developed PoC app was installed on three LG K10 smartphones with Android operational system version 6.

The TP–Link TL–WR542G router was used to provide wireless network using protocol IEEE 802.11g (Wi-Fi) considering two scenarios:

1. LAN network: the tracker and the three cell phones are connected to the same local network without Internet.
2. Network with NAT: the tracker and cell phones are connected to two different wireless networks using protocol IEEE 802.11g.

For scenario 1, the functionality tests were satisfactory. Initially, the app established connections, exchanged messages end-to-end encrypted, and allowed verification and signing of digital certificates by the PGP web of trust. After the modifications for PoC with the insertion of the hash chain, the sending and recording of the messages must obey the chain creation rules with the due checks of the hashes. In tests, the app has proved to be consistent and the checks are performed correctly.

For scenario 2, the connection between the devices has not been established as expected. It will be necessary to check how the connection is configured in the case of IPv4 NAT networks, and it is also interesting to consider the use of IPv6 to establish the connection. Other interesting future test is to use mobile communication via 3G, 4G/LTE, and 5G, also test with same and different carriers.

### 8.1.2 Python PoC

Based on the Slyther and use cases raised in Section 5.1, the following tests were performed at the Python PoC:

1. User does his first registration in the app;
2. User establishes a connection with another user in the network;
3. Users exchange messages;
4. Compare the construction of the hash chain of the conversations between the users;  
and
5. Export the hash chain in a JSON file.

For these tests, we run two GNU/Linux instances of containers in Docker<sup>5</sup> to perform two users, and the functionality tests were satisfactory. The chat established connections

---

<sup>5</sup><https://www.docker.com/>

and exchanged messages end-to-end encrypted. After the insertion of the hash chain in the Slyther chat system, the sending and recording of the messages must obey the hash chain creation rules with the due checks of the hashes. We also inserted the **selective disclosure** and **agree to disagree**, and the chat has proved to be consistent and the checks are performed correctly.

## 8.2 Efficiency analysis: Android benchmarks

In addition to the latency normally found in any network connection, a critical part that determines the latency time of the proposed app’s network is the search time in the DHT network. In this work, the Kademlia algorithm was chosen due to its high performance and scalability [CHÁVEZ; CORTÉS; GUERRERO, 2015]. In particular, searching for any key in Kademlia involves querying (iterative) to  $O(\lg n)$  nodes, where  $n$  is the full size of the network. Numerical values were not measured for this case due to a lack of time and resources such as a considerable scalable number of devices and emulating systems.

To estimate the efficiency of the IM chat system with all the modifications that we proposed, we created a benchmark for each security function (SHA3, ECDH, ECDSA, PGP), and these functions were executed **500 times** at four different devices. In Table 7 we described the devices used, their commercial name, the release year, the Android version in the devices tested, and RAM memory and CPU resources.

Table 7: Specification of the test devices

Device name	Android	RAM memory	CPU
Samsung Galaxy J2 Prime (2016)	6	1.5 GB	1.4 GHz Quad-Core
How HT-705 tablet (2018)	7.1	1 GB	1.2 GHz Quad-Core
Xiaomi Redmi Note 7 (2019)	10	4 GB	2.20 GHz Octa-Core
Samsung Galaxy A22 (2021)	12	4 GB	2 GHz Octa-Core

Source: Authors.

Tables 8 and 9 present the time statistics from ECDSA signature and verification tests respectively. As expected, the verification function takes longer than the signature, and on newer devices with better resources, the absolute time is minor and the time difference between signature and verification gets bigger.



Table 8: ECDSA signature benchmark

Device	Mean (ms)	Standard deviation (ms)	Median (ms)
Samsung Galaxy J2 Prime	6.738	0.549	7.000
How HT-705 tablet	10.472	4.148	9.000
Xiaomi Redmi Note 7	0.498	0.527	0.000
Samsung Galaxy A22	0.386	0.487	0.000

Source: Authors.

Table 9: ECDSA verification benchmark

Device	Mean (ms)	Standard deviation (ms)	Median (ms)
Samsung Galaxy J2 Prime	7.594	0.722	8.000
How HT-705 tablet	11.576	4.981	10.000
Xiaomi Redmi Note 7	0.780	0.464	1.000
Samsung Galaxy A22	0.704	0.456	1.000

Source: Authors.

Table 10 presents the statistics from ECDH key exchange times. As we can observe, on newer devices the times are close to 1 millisecond, interfering very little with the user experience.

Table 10: ECDH key exchange benchmark

Device	Mean (ms)	Standard deviation (ms)	Median (ms)
Samsung Galaxy J2 Prime	13.884	2.552	13.000
How HT-705 tablet	15.320	14.653	12.000
Xiaomi Redmi Note 7	1.578	1.683	1.000
Samsung Galaxy A22	1.202	1.598	1.000

Source: Authors.

Tables 11, 12 and 13 present mean, standard deviation and median statistics, respectively, related to the 256 bits SHA3 hash function. In these tests, we executed the SHA3

with 6 different sizes of inputs 500 times each. In our system, the hash input will be the message exchanged in the IM chat, so we tested messages with 5, 50, 100, 200, 500, and 1000 characters. As we expected, the message’s size does not interfere significantly and the times are very short. A highlight for messages with 1000 characters, which is considered a very long message and which will rarely be sent in message exchange systems by smartphone apps, however, the measured times are reasonable in terms of usability. A comparison, on the social network Twitter, the publication’s characters limit is 280 characters<sup>6</sup>.

Table 11: SHA3-256 benchmark mean times (ms)

<b>Message’s size (#characters)</b>	Samsung Galaxy J2 Prime	How HT-705 tablet	Xiaomi Redmi Note 7	Samsung Galaxy A22
5	0.310	1.018	0.068	0.060
50	0.254	0.298	0.020	0.024
100	0.272	0.238	0.024	0.032
200	0.470	0.340	0.042	0.024
500	0.904	0.742	0.058	0.042
1000	1.822	1.066	0.088	0.072

Source: Authors.

Table 12: SHA3-256 benchmark standard deviation times (ms)

<b>Message’s size (#characters)</b>	Samsung Galaxy J2 Prime	How HT-705 tablet	Xiaomi Redmi Note 7	Samsung Galaxy A22
5	0.504	1.554	0.260	0.246
50	0.435	0.503	0.140	0.153
100	0.445	0.426	0.153	0.176
200	0.503	0.474	0.201	0.153
500	0.308	1.586	0.234	0.201
1000	0.463	2.095	0.283	0.258

Source: Authors.

<sup>6</sup>Counting characters when composing Tweets. <<https://developer.twitter.com/en/docs/counting-characters>>

Table 13: SHA3-256 benchmark median times (ms)

<b>Message's size (#characters)</b>	Samsung Galaxy J2 Prime	How HT-705 tablet	Xiaomi Redmi Note 7	Samsung Galaxy A22
5	0.000	1.000	0.000	0.000
50	0.000	0.000	0.000	0.000
100	0.000	0.000	0.000	0.000
200	0.000	0.000	0.000	0.000
500	1.000	1.000	0.000	0.000
1000	2.000	1.000	0.000	0.000

Source: Authors.

Table 14 presents the statistics related to the signature used in the PGP web of trust tested in the four devices.

Table 14: PGP signature benchmark

<b>Device</b>	<b>Mean (ms)</b>	<b>Standard deviation (ms)</b>	<b>Median (ms)</b>
Samsung Galaxy J2 Prime	88.806	3.617	87.000
How HT-705 tablet	57.724	2.649	58.000
Xiaomi Redmi Note 7	22.062	1.721	21.000
Samsung Galaxy A22	22.620	0.648	23.000

Source: Authors.

Tables 15, 16, 17 and 18 present the statistics related to the encryption using PGP in the four devices tested. We measured 500 times the signature in the same 6 different sizes of messages used in the SHA3 tests. In this case, encryption is important due to the end-to-end encryption in the IM chat, so the message size can be relevant. As we can observe, the larger the message, the longer the encryption time, but the time difference is still very small.

Table 15: PGP encryption benchmark in Samsung Galaxy J2 Prime

Message's size (#characters)	Mean (ms)	Standard deviation (ms)	Median (ms)
5	4.198	0.956	4.000
50	4.240	0.531	4.000
100	4.258	0.707	4.000
200	4.386	0.530	4.000
500	4.740	0.544	5.000
1000	5.292	0.561	5.000

Source: Authors.

Table 16: PGP encryption benchmark in How HT-705 tablet

Message's size (#characters)	Mean (ms)	Standard deviation (ms)	Median (ms)
5	10.706	4.391	9.000
50	8.574	1.117	8.000
100	9.198	3.074	9.000
200	8.786	0.976	9.000
500	9.570	2.259	9.000
1000	9.866	0.963	10.000

Source: Authors.

Table 17: PGP encryption benchmark in Xiaomi Redmi Note 7

Message's size (#characters)	Mean (ms)	Standard deviation (ms)	Median (ms)
5	3.926	0.986	4.000
50	5.326	1.563	6.000
100	6.056	0.530	6.000
200	5.952	0.523	6.000
500	6.044	0.535	6.000
1000	6.156	0.521	6.000

Source: Authors.

Table 18: PGP encryption benchmark in Samsung Galaxy A22

Message's size (#characters)	Mean (ms)	Standard deviation (ms)	Median (ms)
5	2.584	0.927	2.000
50	2.870	1.202	2.000
100	4.072	0.774	4.000
200	4.036	0.771	4.000
500	4.100	0.833	4.000
1000	4.116	0.794	4.000

Source: Authors.

Tables 19, 20, 21 and 22 present the statistics related to the decryption using PGP in the four devices tested. Being the reverse process of the encryption, we performed the same test for the decryption. Same to encryption, the decryption times increase a little according to the message size increase.

Table 19: PGP decryption benchmark in Samsung Galaxy J2 Prime

Message's size (#characters)	Mean (ms)	Standard deviation (ms)	Median (ms)
5	89.952	4.850	88.000
50	89.710	3.870	88.000
100	90.354	4.637	88.000
200	90.022	3.898	88.000
500	90.980	4.306	89.000
1000	92.046	4.367	90.000

Source: Authors.

Table 20: PGP decryption benchmark in How HT-705 tablet

Message's size (#characters)	Mean (ms)	Standard deviation (ms)	Median (ms)
5	66.982	9.949	65.000
50	60.966	3.807	61.000
100	63.948	12.076	62.000
200	61.518	3.372	61.000
500	64.140	9.775	62.000
1000	63.142	3.726	63.000

Source: Authors.

Table 21: PGP decryption benchmark in Xiaomi Redmi Note 7

Message's size (#characters)	Mean (ms)	Standard deviation (ms)	Median (ms)
5	13.160	1.796	13.000
50	19.284	5.247	22.000
100	22.726	1.229	22.000
200	22.714	1.210	22.000
500	22.894	1.282	22.000
1000	23.214	1.452	23.000

Source: Authors.

Table 22: PGP decryption benchmark in Samsung Galaxy A22

Message's size (#characters)	Mean (ms)	Standard deviation (ms)	Median (ms)
5	13.140	2.317	13.000
50	15.808	5.691	12.500
100	24.448	1.571	24.000
200	24.392	1.626	24.000
500	24.464	1.695	24.000
1000	24.612	1.557	24.000

Source: Authors.

If we consider all the processes to send and receive a message in the proposed system, we have the sum of times (ECDH exchange + 3 x SHA3-256 + PGP encryption + ECDSA sign) to the sending and the sum of times (ECDH exchange + 3 x SHA3-256 + ECDSA verify + PGP decryption) to the receiving. For calculation, consider the newest device tested (Samsung Galaxy A22) and a message with 100 characters. If we add the respective average times, we have about 5.756ms to the sending and 26.450ms to the receiving, generating a total time of 32.206ms. In the literature, [KEATES, 2016; FUNK et al., 2020] says that the better response time is  $\approx 250$ ms, and [FUNK et al., 2020] considers that the most accepted time range is between 0s and 2s. Seeing out benchmark times, we conclude that the system proves to be efficient enough for the standards in the literature.

### 8.3 Integrity and auditing testing of conversations

Based on Section 6.2.1, we tested each scenario in the Python PoC. First, we created a chat presented in Figure 23 that generate the hash chain presented in the followed Figure 24.

Figure 23: Chat generated in the Python PoC

```
docker exec -it 23968a23c00ad0943584f5f13fbd7e4a3a2e078fbd7b130e0bce7575aff7ba881 /bin/sh
root@23968a23c00a:/slyther# python3 slyther
////////////////////
// s l y t h e r //
////////////////////
Please log in...
Password:
Login successful.

Enter a command...
oc  -> Open Conversation
ec  -> Edit Conversation
nc  -> New/Update Contact
lc  -> List Contacts
dc  -> Delete Contact
fp  -> Display Fingerprint
pc  -> Export hashchain
c   -> Clear Screen
h   -> Display help
q   -> Quit slyther

> oc
----- VIEW CONVERSATION -----
felipe      >  it is ok                    <  09/23/22 07:26PM

Contact Name: felipe
----- CONVERSATION -----
09/23/22 04:33PM me: hello
09/23/22 04:34PM felipe: hi
09/23/22 04:34PM me: test 1
09/23/22 04:34PM me: test 2
09/23/22 04:34PM me: test 3
09/23/22 04:34PM felipe: ok 1
09/23/22 04:35PM felipe: ok 2
09/23/22 04:35PM me: ok 3?
09/23/22 04:35PM felipe: ok 3
09/23/22 04:35PM felipe: finish
09/23/22 07:26PM felipe: new test
09/23/22 07:26PM me: it is ok

Message: ^C

>
```

Source: Authors.



Figure 24: Hash chain generated in the Python PoC

```

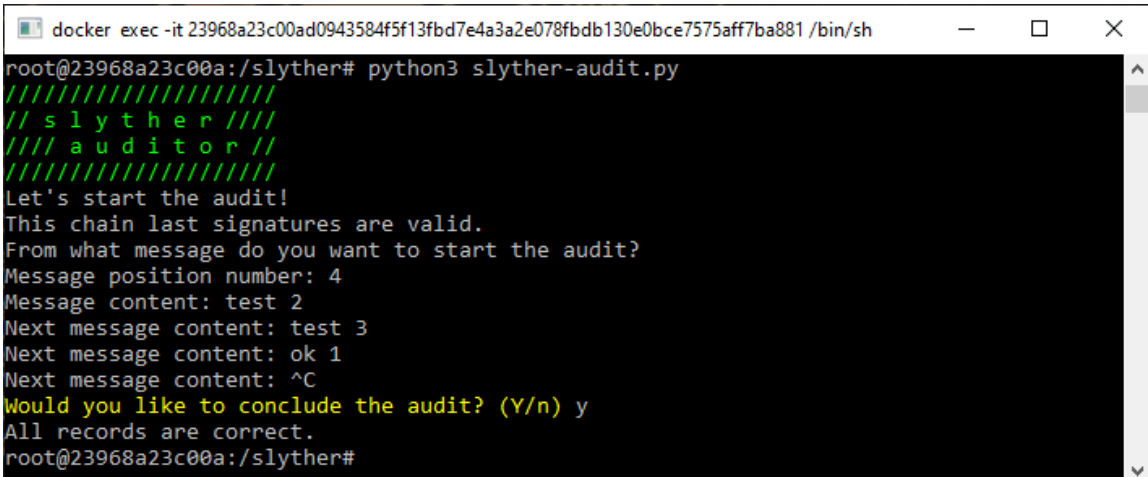
1 | "hashchain":
2 | [{"prev_hash": "0000000000000000000000000000000000000000000000000000000000000000",
3 | "message_x": "f293f4a18b8716e95a8549bb4dfdb7bed1b84a5e6e4e2dbfebad40ae652fd54b"},
4 | {"prev_hash": "66a3ceea234052f7371afcc435ed0c2f8f140cb46a3e1597ece822fa622db652",
5 | "message_x": "6a3dc382d99661b09350ee96d0439be3059e954da428353d5851b3435ed27357"},
6 | {"prev_hash": "c3fec78b12e4ecb233dafcc468f0137af361dcc1cc0ad0cd6016742e7a98f512",
7 | "message_x": "9696b0fb884a64297f13a7b2b4f8e1cfd9d36db43b1554398d240d0f56c8717"},
8 | {"prev_hash": "dc4dbbc8072c1a0c8f069ce1e58ea6b81775016098187f7d67be6e6e511c769f",
9 | "message_x": "520628d290159d2636f14f80413146a7eb7153b67b7c23522a051e048ccb083f"},
10 | {"prev_hash": "93078edec72d02a5e105fb6eef7c00d225e65773c60405a1d3028637fb4a74f5",
11 | "message_x": "63248ad8d993555566f181dcc95e09d3b0c90ad4737eebc5e21b3240be7a991e"},
12 | {"prev_hash": "4c5f7c7b14ddd7b8d13a5ed355942751d52fb3670f8027d5b16dd69f760c2aa6",
13 | "message_x": "fe57855610f529cc1f80d6ca6bacbf3c564bf0dd54c2f8614786876784bd1812"},
14 | {"prev_hash": "a09aa01041917786ba3588b74a6c20559f057fd1fdc9dd904eddd4c345df7da8",
15 | "message_x": "d1ac1e8e5227f19cb4a2484204abfd149de0ff1515866e920891b0637157e982"},
16 | {"prev_hash": "5d3c85775cb94d0f8519887b8e61920bdc11995165296f2b8cec756f56bec644",
17 | "message_x": "db74bf6ac67ea609bece730ee56f8ed6fbfdca4b3982e5a44e21a36c16add970"},
18 | {"prev_hash": "30bc80357ae476ba32bc2960c25ee88dd8bf1c0b43b57251358f608d18e851ec",
19 | "message_x": "02b09ca3111281d4c86ede4ad3e53e87cd42b3a0696fa5f36f123222c2dd2a52"},
20 | {"prev_hash": "1def4a1a756f09165a5139b372a7afce98593d34d260a9233a501831a2f9b862",
21 | "message_x": "15994d41881ab59643763c616d7bb19d4b591df2ed51f6c8469ab1b8e8a0b1ce"}]}

```

Source: Authors.

We developed an independent script to validate and audit a hash chain inputted. This script is also available in the project's GitHub at <<https://github.com/Erina-chan/hashchat/blob/main/slyther-audit.py>>. The script checks the chain's last signature then it starts to ask from what message it will start to audit and what is this message's content. With this data, the script calculates and compares it with the information in the hash chain received, and this process repeats until the end of the chain. Figure 25 presents a partial audit with correct information.

Figure 25: Auditing the chat with correct messages



```

docker exec -it 23968a23c00ad0943584f5f13fbd7e4a3a2e078fbd130e0bce7575aff7ba881 /bin/sh
root@23968a23c00a:/slyther# python3 slyther-audit.py
////////////////////
// slyther //
/// auditor //
////////////////////
Let's start the audit!
This chain last signatures are valid.
From what message do you want to start the audit?
Message position number: 4
Message content: test 2
Next message content: test 3
Next message content: ok 1
Next message content: ^C
Would you like to conclude the audit? (Y/n) y
All records are correct.
root@23968a23c00a:/slyther#

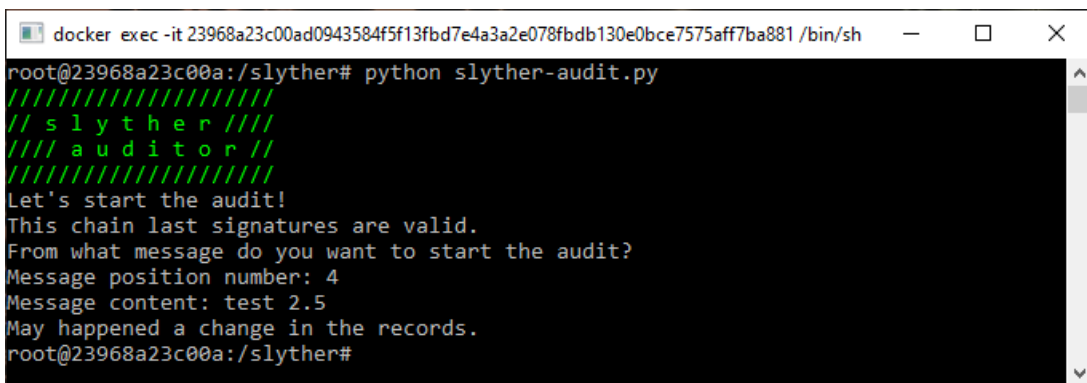
```

Source: Authors.

Executing the same script in the scenarios described in Section 6.2.1, the results are:

- *Editing message content*: For this test, we just informed a different message's content. The system expected `test 2`, but we inputted `test 2.5`. As you can see in Figure 26, the difference with the hash chain record is identified and the script ends stating that it has identified an inconsistency.

Figure 26: Auditing the chat with a different message



```

docker exec -it 23968a23c00ad0943584f5f13fbd7e4a3a2e078fbd130e0bce7575aff7ba881 /bin/sh
root@23968a23c00a:/slyther# python slyther-audit.py
////////////////////
// slyther //
/// auditor ///
////////////////////
Let's start the audit!
This chain last signatures are valid.
From what message do you want to start the audit?
Message position number: 4
Message content: test 2.5
May happened a change in the records.
root@23968a23c00a:/slyther#

```

Source: Authors.

- *Changing the messages' order*: For this test, we changed the fourth message (`test 2`) with the fifth message (`test 3`) position. Figure 27 presents the hash chain used to audit with the fourth and the fifth registers' position changed.

Figure 27: Hash chain modified: changing the fourth and the fifth registers

```

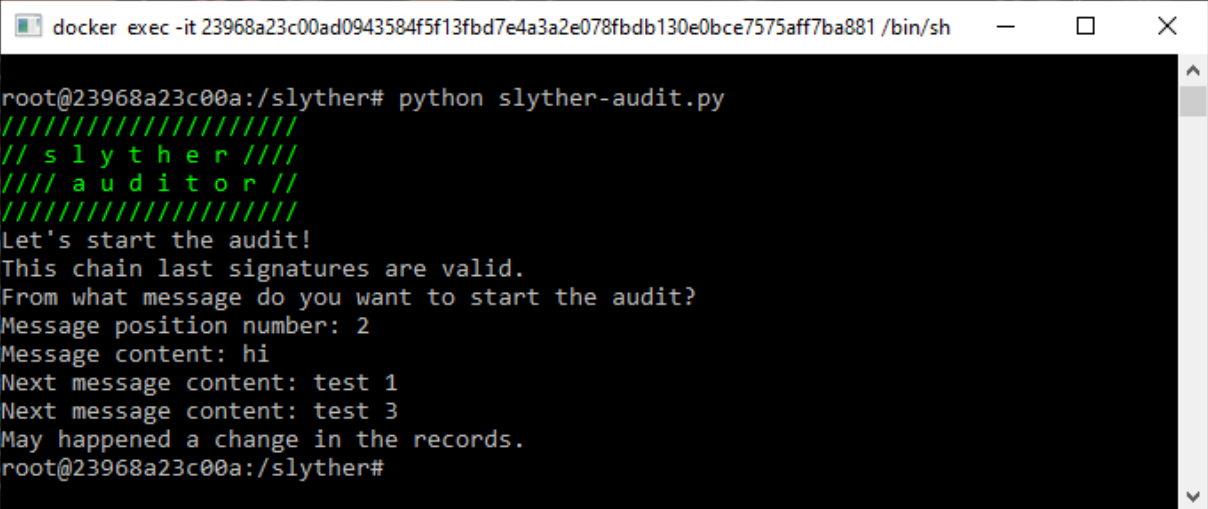
1  "hashchain":
2  [{"prev_hash": "0000000000000000000000000000000000000000000000000000000000000000",
3  "message_x": "f293f4a18b8716e95a8549bb4dfdb7bed1b84a5e6e4e2dbfabad40ae652fd54b"},
4  {"prev_hash": "66a3ceea234052f7371afcc435ed0c2f8f140cb46a3e1597ece822fa622db652"},
5  "message_x": "6a3dc382d99661b09350ee96d0439be3059e954da428353d5851b3435ed27357"},
6  {"prev_hash": "c3fec78b12e4ecb233dafcc468f0137af361dcc1cc0ad0c6016742e7a98f512"},
7  "message_x": "9696b0fb884a64297f13a7b2b4f8e1cfdd9d36db43b1554398d240d0f56c8717"},
8  {"prev_hash": "93078edec72d02a5e105fb6eef7c00d225e65773c60405a1d3028637fb4a74f5"},
9  "message_x": "63248ad8d993555566f181dcc95e09d3b0c90ad4737eebc5e21b3240be7a991e"},
10 {"prev_hash": "dc4dbbc8072c1a0c8f069ce1e58ea6b81775016098187f7d67be6e6e511c769f"},
11 "message_x": "520628d290159d2636f14f80413146a7eb7153b67b7c23522a051e048ccb083f"},
12 {"prev_hash": "4c5f7c7b14ddd7b8d13a5ed355942751d52fb3670f8027d5b16dd69f760c2aa6"},
13 "message_x": "fe57855610f529cc1f80d6ca6bacbf3c564bf0dd54c2f8614786876784bd1812"},
14 {"prev_hash": "a09aa01041917786ba3588b74a6c20559f057fd1fdc9dd904eddd4c345dfda8"},
15 "message_x": "d1ac1e8e5227f19cb4a2484204abfd149de0ff1515866e920891b0637157e982"},
16 {"prev_hash": "5d3c85775cb94d0f8519887b8e61920bdc11995165296f2b8cec756f56bec644"},
17 "message_x": "db74bf6ac67ea609bece730ee56f8ed6fbfdca4b3982e5a44e21a36c16add970"},
18 {"prev_hash": "30bc80357ae476ba32bc2960c25ee88dd8bf1c0b43b57251358f608d18e851ec"},
19 "message_x": "02b09ca3111281d4c86ede4ad3e87cd42b3a0696fa5f36f123222c2dd2a52"},
20 {"prev_hash": "1def4a1a756f09165a5139b372a7afce98593d34d260a9233a501831a2f9b862"},
21 "message_x": "15994d41881ab59643763c616d7bb19d4b591df2ed51f6c8469ab1b8e8a0b1ce"}]}

```

Source: Authors.

As you can see in Figure 28, we informed correctly the content `test 3` in the fourth position, however, when the system compares the previous block's hash calculated with the expected, it identifies an inconsistency.

Figure 28: Auditing the chat with message order changed



```

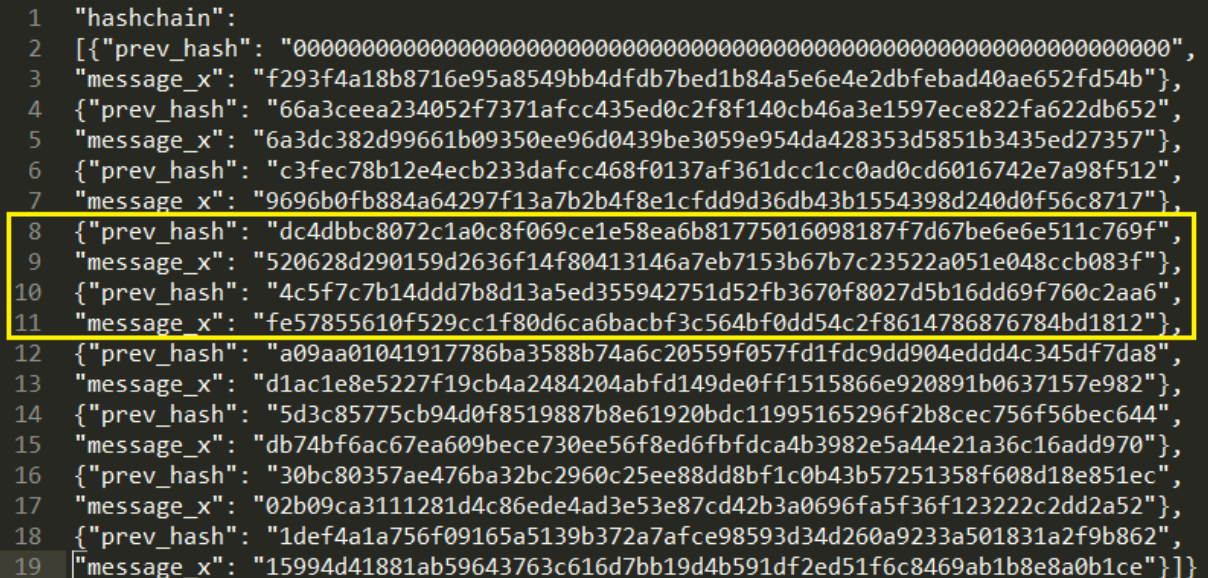
docker exec -it 23968a23c00ad0943584f5f13fbd7e4a3a2e078fbd130e0bce7575aff7ba881 /bin/sh
root@23968a23c00a:/slyther# python slyther-audit.py
////////////////////
// slyther //
/// auditor //
////////////////////
Let's start the audit!
This chain last signatures are valid.
From what message do you want to start the audit?
Message position number: 2
Message content: hi
Next message content: test 1
Next message content: test 3
May happened a change in the records.
root@23968a23c00a:/slyther#

```

Source: Authors.

- *Removing messages*: For this test, we deleted the fourth message (`test 2`). Figure 29 presents the hash chain used to audit where the fourth register was deleted.

Figure 29: Hash chain modified: deleting the fourth register



```

1  "hashchain":
2  [{"prev_hash": "0000000000000000000000000000000000000000000000000000000000000000",
3  "message_x": "f293f4a18b8716e95a8549bb4dfdb7bed1b84a5e6e4e2dbfebad40ae652fd54b"},
4  {"prev_hash": "66a3ceea234052f7371afcc435ed0c2f8f140cb46a3e1597ece822fa622db652",
5  "message_x": "6a3dc382d99661b09350ee96d0439be3059e954da428353d5851b3435ed27357"},
6  {"prev_hash": "c3fec78b12e4ecb233dafcc468f0137af361dcc1cc0ad0cd6016742e7a98f512",
7  "message_x": "9696b0fb884a64297f13a7b2b4f8e1cfd9d36db43b1554398d240d0f56c8717"},
8  {"prev_hash": "dc4dbbc8072c1a0c8f069ce1e58ea6b81775016098187f7d67be6e6e511c769f",
9  "message_x": "520628d290159d2636f14f80413146a7eb7153b67b7c23522a051e048ccb083f"},
10 {"prev_hash": "4c5f7c7b14ddd7b8d13a5ed355942751d52fb3670f8027d5b16dd69f760c2aa6",
11 "message_x": "fe57855610f529cc1f80d6ca6bacbf3c564bf0dd54c2f8614786876784bd1812"},
12 {"prev_hash": "a09aa01041917786ba3588b74a6c20559f057fd1fdc9dd904eddd4c345df7da8",
13 "message_x": "d1ac1e8e5227f19cb4a2484204abfd149de0ff1515866e920891b0637157e982"},
14 {"prev_hash": "5d3c85775cb94d0f8519887b8e61920bdc11995165296f2b8cec756f56bec644",
15 "message_x": "db74bf6ac67ea609bece730ee56f8ed6fbfdca4b3982e5a44e21a36c16add970"},
16 {"prev_hash": "30bc80357ae476ba32bc2960c25ee88dd8bf1c0b43b57251358f608d18e851ec",
17 "message_x": "02b09ca3111281d4c86ede4ad3e53e87cd42b3a0696fa5f36f123222c2dd2a52"},
18 {"prev_hash": "1def4a1a756f09165a5139b372a7afce98593d34d260a9233a501831a2f9b862",
19 "message_x": "15994d41881ab59643763c616d7bb19d4b591df2ed51f6c8469ab1b8e8a0b1ce"}]]

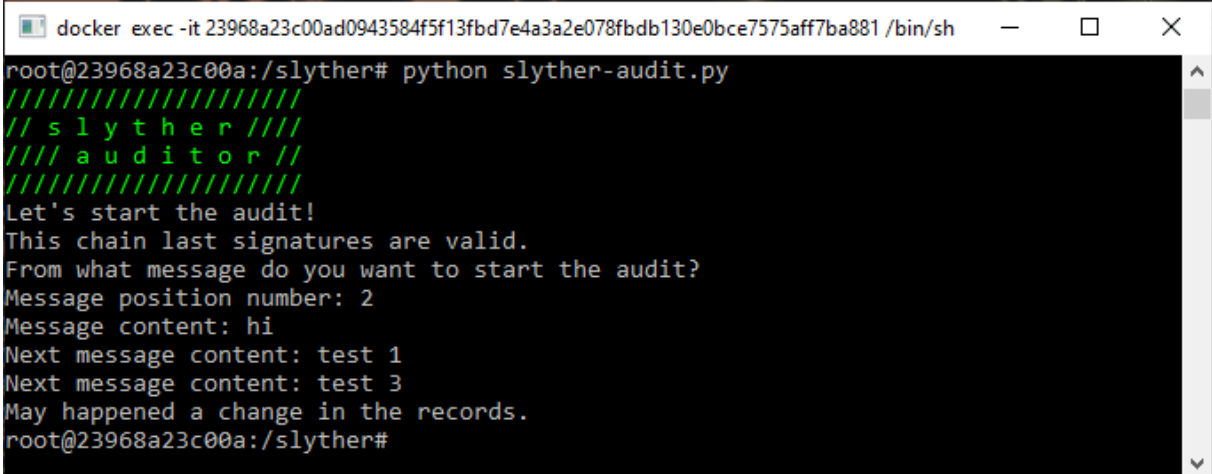
```

Source: Authors.

As you can see in Figure 30, we informed correctly the messages' content without

test 2, nevertheless when the system compares the previous block's hash calculated with the expected, it identifies an inconsistency.

Figure 30: Auditing the chat with message deleted



```

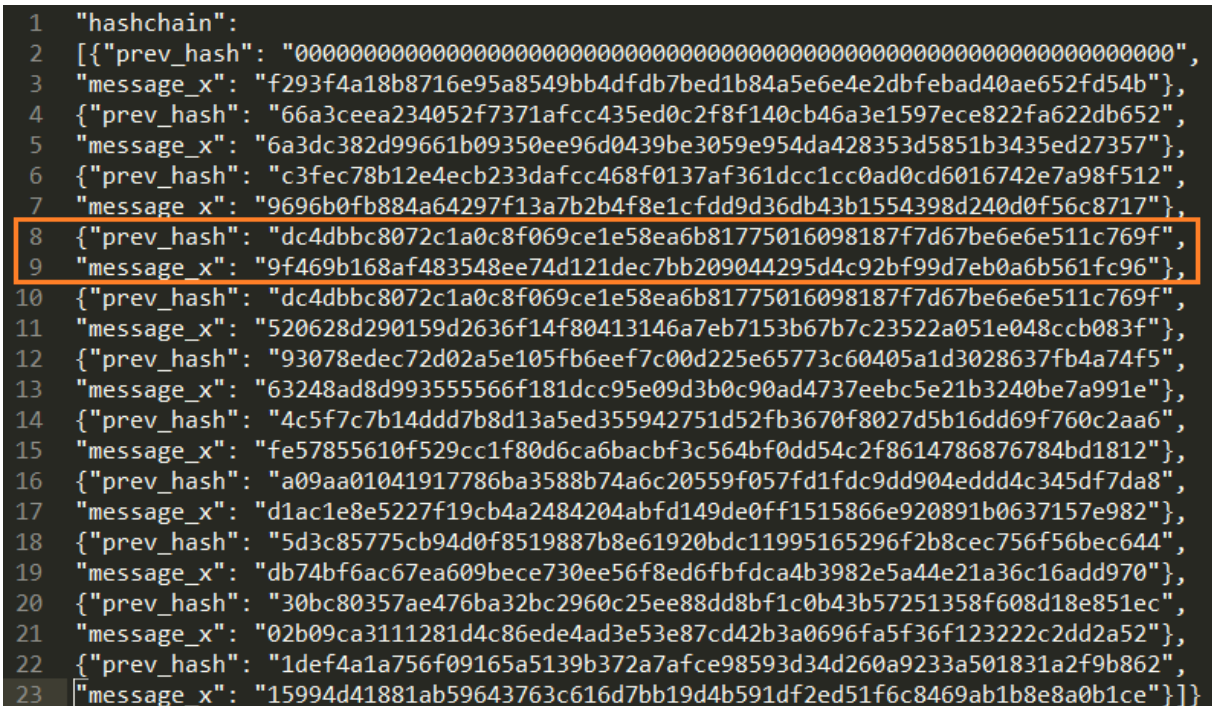
root@23968a23c00a:/slyther# python slyther-audit.py
////////////////////
// slyther //
//// auditor //
////////////////////
Let's start the audit!
This chain last signatures are valid.
From what message do you want to start the audit?
Message position number: 2
Message content: hi
Next message content: test 1
Next message 3 content: test 3
May happened a change in the records.
root@23968a23c00a:/slyther#

```

Source: Authors.

- *Inserting messages*: For this test, we added in the fourth position a message (new text add). Figure 31 presents the hash chain used to audit where the fourth register was added with the third block's hash and "new text add"'s message\_x code.

Figure 31: Hash chain modified: adding a new the fourth register



```

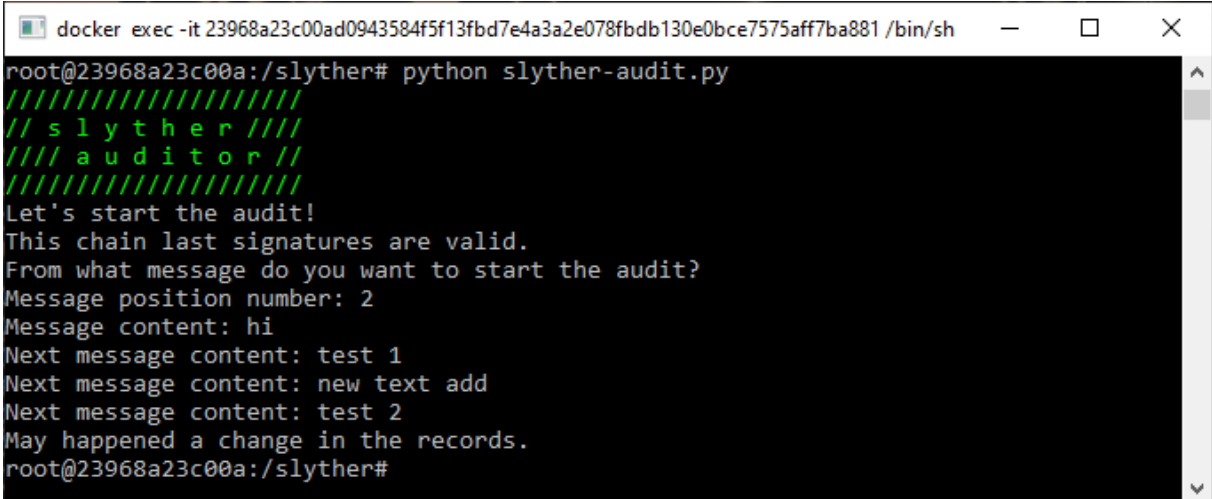
1  "hashchain":
2  [{"prev_hash": "0000000000000000000000000000000000000000000000000000000000000000",
3  "message_x": "f293f4a18b8716e95a8549bb4dfdb7bed1b84a5e6e4e2dbfebad40ae652fd54b"},
4  {"prev_hash": "66a3ceea234052f7371afcc435ed0c2f8f140cb46a3e1597ece822fa622db652",
5  "message_x": "6a3dc382d99661b09350ee96d0439be3059e954da428353d5851b3435ed27357"},
6  {"prev_hash": "c3fec78b12e4ecb233dafcc468f0137af361dcc1cc0ad0cd6016742e7a98f512",
7  "message_x": "9696b0fb884a64297f13a7b2b4f8e1cfd9d36db43b1554398d240d0f56c8717"},
8  {"prev_hash": "dc4dbbc8072c1a0c8f069ce1e58ea6b81775016098187f7d67be6e6e511c769f",
9  "message_x": "9f469b168af483548ee74d121dec7bb209044295d4c92bf99d7eb0a6b561fc96"},
10 {"prev_hash": "dc4dbbc8072c1a0c8f069ce1e58ea6b81775016098187f7d67be6e6e511c769f",
11 "message_x": "520628d290159d2636f14f80413146a7eb7153b67b7c23522a051e048ccb083f"},
12 {"prev_hash": "93078edec72d02a5e105fb6eef7c00d225e65773c60405a1d3028637fb4a74f5",
13 "message_x": "63248ad8d993555566f181dcc95e09d3b0c90ad4737eebc5e21b3240be7a991e"},
14 {"prev_hash": "4c5f7c7b14ddd7b8d13a5ed355942751d52fb3670f8027d5b16dd69f760c2aa6",
15 "message_x": "fe57855610f529cc1f80d6ca6bacbf3c564bf0dd54c2f8614786876784bd1812"},
16 {"prev_hash": "a09aa01041917786ba3588b74a6c20559f057fd1fdc9dd904eddd4c345df7da8",
17 "message_x": "d1ac1e8e5227f19cb4a2484204abfd149de0ff1515866e920891b0637157e982"},
18 {"prev_hash": "5d3c85775cb94d0f8519887b8e61920bdc11995165296f2b8cec756f56bec644",
19 "message_x": "db74bf6ac67ea609bece730ee56f8ed6fbfdca4b3982e5a44e21a36c16add970"},
20 {"prev_hash": "30bc80357ae476ba32bc2960c25ee88dd8bf1c0b43b57251358f608d18e851ec",
21 "message_x": "02b09ca3111281d4c86ede4ad3e53e87cd42b3a0696fa5f36f123222c2dd2a52"},
22 {"prev_hash": "1def4a1a756f09165a5139b372a7afce98593d34d260a9233a501831a2f9b862",
23 ["message_x": "15994d41881ab59643763c616d7bb19d4b591df2ed51f6c8469ab1b8e8a0b1ce"]}

```

Source: Authors.

As you can see in Figure 32, we informed correctly the messages' content, and the new message `new text add` was considered correct. However, when the system verifies the next message (`text 2`) and compares the previous block's hash calculated with the expected, it identifies an inconsistency.

Figure 32: Auditing the chat with message added



```

docker exec -it 23968a23c00ad0943584f5f13fbd7e4a3a2e078fbd130e0bce7575aff7ba881 /bin/sh
root@23968a23c00a:/slyther# python slyther-audit.py
////////////////////
// slyther //
/// auditor //
////////////////////
Let's start the audit!
This chain last signatures are valid.
From what message do you want to start the audit?
Message position number: 2
Message content: hi
Next message content: test 1
Next message content: new text add
Next message content: test 2
May happened a change in the records.
root@23968a23c00a:/slyther#

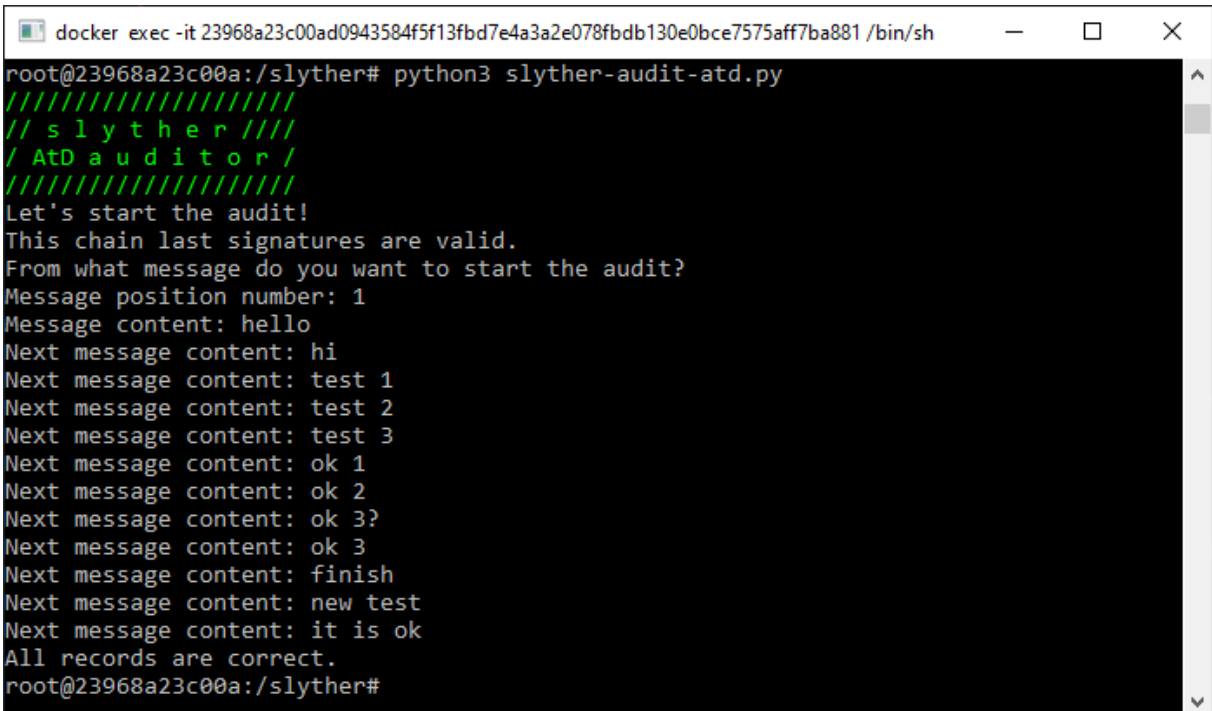
```

Source: Authors.

As we described, during an audit, the system just identifies the possible modification, it does not present the exact modification. For example, as we can see, the Figures 28 and 30 present the same output in the audit, then with just this information, we can not know what was the modification. The only certainty is when the audited chat is correct and complete, and we do not aim to identify what was the modification or what was the original plain text.

In addition to these scenarios, we also tested performing an audit with a collision in the chat. For this test, we developed another independent script to validate and audit a hash chain and this script is also available in the project's GitHub at <https://github.com/Erina-chan/hashchat/blob/main/slyther-audit-atd.py>. This script does the same audit as the previous one and also handles the scenario with the collision. Figure 33 show the audit of the full chat used in the previous tests.

Figure 33: Auditing the full chat with correct messages' content



```

docker exec -it 23968a23c00ad0943584f5f13fbd7e4a3a2e078fbd7b130e0bce7575aff7ba881 /bin/sh
root@23968a23c00a:/slyther# python3 slyther-audit-atd.py
////////////////////
// s l y t h e r //
/ AtD a u d i t o r /
////////////////////
Let's start the audit!
This chain last signatures are valid.
From what message do you want to start the audit?
Message position number: 1
Message content: hello
Next message content: hi
Next message content: test 1
Next message content: test 2
Next message content: test 3
Next message content: ok 1
Next message content: ok 2
Next message content: ok 3?
Next message content: ok 3
Next message content: finish
Next message content: new test
Next message content: it is ok
All records are correct.
root@23968a23c00a:/slyther#

```

Source: Authors.

To generate the chat with collision we used the script available in the project's GitHub at <https://github.com/Erina-chan/hashchat/blob/main/atd-script.py>. Figure 34 shows the chat created, as we can see, the collision is present in the fourth and fifth positions, and Figure 35 presents this chat's respective hash chain. Due to collision, the hash chain presents a “agree-to-disagree (AtD)” block with the hashes of each block in the sequence seen by each interlocutor.

Figure 34: Chat generated with collision in the Python PoC

```

docker exec -it 23968a23c00ad0943584f5f13fbd7e4a3a2e078fbd130e0bce7575aff7ba881 /bin/sh
root@23968a23c00a:/slyther# python3 atd-script.py
////////////////////
// s l y t h e r //
//// c h a t + AtD //
////////////////////
My password:
Contact password:
Let's start the chat!
Message content: hello
You are the sender? (Y/n) y
Next message content: hi
You are the sender? (Y/n) n
This message's position is number: 2
Next message content: are you ok?
You are the sender? (Y/n) n
This message's position is number: 3
Next message content: yes, fine
You are the sender? (Y/n) y
This message's position is number: 4
Next message content: let's play?
You are the sender? (Y/n) y
This message's position is number: 5
Next message content: now
You are the sender? (Y/n) y
This message's position is number: 6
Next message content: still working?
You are the sender? (Y/n) n
This message's position is number: 4

Colission detected. Do you want to finish the collision (Y/n) n
Next message content in the collision: could you help me?
This message's position is number: 5

Colission detected. Do you want to finish the collision (Y/n) y
My next message after collision end: sure
Next message content: what is the problem?
You are the sender? (Y/n) y
This message's position is number: 12
Next message content: I will send in your email
You are the sender? (Y/n) n
This message's position is number: 13
Next message content: ^C
Do you like to finish atd-script? (Y/n) y
root@23968a23c00a:/slyther#

```

Source: Authors.

Figure 35: Hash chain generated from chat with collision in the Python PoC

```

documents\docker-works\atd-test\hashchain.json - Sublime Text (UNREGISTERED)
Find View Goto Tools Project Preferences Help
hashchain.json
1 { "my_last_sign_position": 12,
2   "contact_last_sign_position": 13,
3   "hashchain": [{
4     "prev_hash": "0000000000000000000000000000000000000000000000000000000000000000",
5     "message_x": "6c8d8988661778f61b47ffd7d4b65f4e26677ae39ba4e395df32d54320df9a8f",
6   }, {"prev_hash": "d8bb40910cafcd7bd884ce624fb44ead13fe7e6f875a1393853317dbccca992eb",
7     "message_x": "ed103b75b0c1e905161a1a81abbd7dba810e1e303d1231fbd115e93473d6d18c",
8   }, {"prev_hash": "42a2fdb0923c96d535c1bdfc3bad032046f96354b1ed8c95766186a5c2e2781e",
9     "message_x": "e6b497c8130107aa3aff56848de2751b407ac568fe397e72f801d6e785e4bfff",
10    }, {"prev_hash": "7724068fab048165e4bfc2c2fb6b620e430e0f23203eac46e0ab5c9c8434d5bfff",
11      "message_x": "502035ee8da53fcf7cbb54b8bbfe5306d3af526b376a23bca2d8e25a6f214314",
12    }, {"prev_hash": "623df53e2916e166da7c9040ba4c1cdaf5f14f121ee73c5663e9c2d78f7f1c2a",
13      "message_x": "6bf9fa6fad3078bce3b811d1c393228014dac2603a3f99b1899574621e5cf78b",
14    }, {"prev_hash": "4985e0ee53b9772013eea9349cb8d837ee4a1599c22dbacebcacf808fa0c5007f",
15      "message_x": "6c7ac148757847be3ee7cc9ecd4763e30977d9500355a40ca9c463e7762e452e",
16    }, {"prev_hash": "7724068fab048165e4bfc2c2fb6b620e430e0f23203eac46e0ab5c9c8434d5bfff",
17      "message_x": "881f7a515ae998599d8d34dec54b839e65faeb929f9065861ddb0caac60a1c8",
18    }, {"prev_hash": "e8e750dbbd6087d9d219fdeb2f5e5b185b7c58884d571c23d66e468b5fc4f7bf",
19      "message_x": "81bcc300562e77d64e4c08f7dbbcfccd1147dad7dcc6d5f0c9ddc3fdd30b4761",
20    }, {"collision_start": 4,
21      "sequency": [
22        "7724068fab048165e4bfc2c2fb6b620e430e0f23203eac46e0ab5c9c8434d5bfff",
23        "623df53e2916e166da7c9040ba4c1cdaf5f14f121ee73c5663e9c2d78f7f1c2a",
24        "4985e0ee53b9772013eea9349cb8d837ee4a1599c22dbacebcacf808fa0c5007f",
25        "431e2f2f28dd3b9cb42ba952e0cea3b014ada24af0ca535362be360a5e9fd14f",
26        "e8e750dbbd6087d9d219fdeb2f5e5b185b7c58884d571c23d66e468b5fc4f7bf",
27        "651df7577e73846305bce7b92b1070f8a10d100e5b8b73d9dca17f5becf5f7de"]
28    }, {"collision_start": 4,
29      "sequency": [
30        "7724068fab048165e4bfc2c2fb6b620e430e0f23203eac46e0ab5c9c8434d5bfff",
31        "e8e750dbbd6087d9d219fdeb2f5e5b185b7c58884d571c23d66e468b5fc4f7bf",
32        "651df7577e73846305bce7b92b1070f8a10d100e5b8b73d9dca17f5becf5f7de",
33        "623df53e2916e166da7c9040ba4c1cdaf5f14f121ee73c5663e9c2d78f7f1c2a",
34        "4985e0ee53b9772013eea9349cb8d837ee4a1599c22dbacebcacf808fa0c5007f",
35        "431e2f2f28dd3b9cb42ba952e0cea3b014ada24af0ca535362be360a5e9fd14f"]
36    }, {"prev_hash": "88ecfa71812b15bee2cd2b2a8f321db2ed6ee80b58e79e896b1c08fa36744501",
37      "message_x": "5d89ff794fdd0e0bad808355f5afdcffc4b429a39bbbd988e4dadcd8f997819f2",
38    }, {"prev_hash": "2c73e355e449dda3d85d0132c2cbce00e56086014449ce13eea3962b58ce2373",
39      "message_x": "670788b136d282f69ed265d20dbd259d263664470181cbcad4d5596bf34bb7fe",
40    }, {"prev_hash": "65eedca9186610c18150f7efdbdecc4adf98f79c3e98f1ad6dcab5aa52c0ad5c",
41      "message_x": "6feece12ca0f6535cbbf5733769562add7dcd3c0fde4dd18558dc8d3349aead3",
42    }
43  ]
44 }

```

Source: Authors.

So in Figure 36 was realized a partial audit starting from the fifth message “let’s play”. First, we executed the test with the correct messages’ content (“let’s play” and “now”), these data were validated and all the hash chain remainder also were checked and returned “All records are correct”. Second, we ran the same test but with the wrong message’s content (“let’s play” and “fine”), the script identified the inconsistency and returned “May happened a change in the records”.



Figure 36: Partial audit tests from chat with collision

```

docker exec -it 23968a23c00ad0943584f5f13fbd7e4a3a2e078fbd130e0bce7575aff7...
root@23968a23c00a:/slyther# python3 slyther-audit-atd.py
////////////////////
// s l y t h e r ///
/ A t D a u d i t o r /
////////////////////
Let's start the audit!
This chain last signatures are valid.
From what message do you want to start the audit?
Message position number: 5
Message content: let's play?
Next message content: now
Next message content: ^C
Would you like to conclude the audit? (Y/n) y
All records are correct.
root@23968a23c00a:/slyther# python3 slyther-audit-atd.py
////////////////////
// s l y t h e r ///
/ A t D a u d i t o r /
////////////////////
Let's start the audit!
This chain last signatures are valid.
From what message do you want to start the audit?
Message position number: 5
Message content: let's play?
Next message content: fine
May happened a change in the records.
root@23968a23c00a:/slyther#

```

Source: Authors.

Next, we realized a partial test starting after the collision ended. From Figure 37 we can see that everything was checked correctly.

Figure 37: Partial audit test starting after collision

```

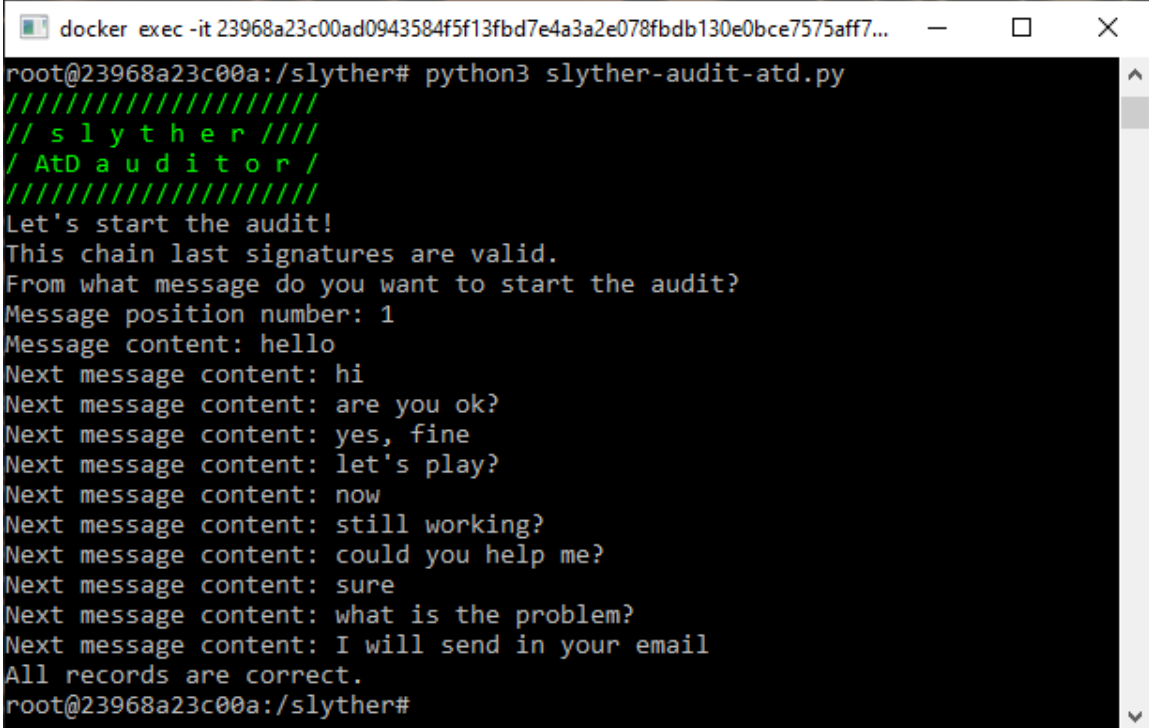
docker exec -it 23968a23c00ad0943584f5f13fbd7e4a3a2e078fbd130e0bce75...
root@23968a23c00a:/slyther# python3 slyther-audit-atd.py
////////////////////
// s l y t h e r ///
/ A t D a u d i t o r /
////////////////////
Let's start the audit!
This chain last signatures are valid.
From what message do you want to start the audit?
Message position number: 11
Message content: sure
Next message content: what is the problem?
Next message content: I will send in your email
All records are correct.
root@23968a23c00a:/slyther#

```

Source: Authors.

And last we checked the full chat with the correct messages' contents, shown in Figure 38.

Figure 38: Auditing full chat with collision



```
docker exec -it 23968a23c00ad0943584f5f13fbd7e4a3a2e078fbd7b130e0bce7575aff7...
root@23968a23c00a:/slyther# python3 slyther-audit-atd.py
////////////////////
// slyther //
/ AtD auditor /
////////////////////
Let's start the audit!
This chain last signatures are valid.
From what message do you want to start the audit?
Message position number: 1
Message content: hello
Next message content: hi
Next message content: are you ok?
Next message content: yes, fine
Next message content: let's play?
Next message content: now
Next message content: still working?
Next message content: could you help me?
Next message content: sure
Next message content: what is the problem?
Next message content: I will send in your email
All records are correct.
root@23968a23c00a:/slyther#
```

Source: Authors.

In this case, we also can just identify that happened some modification, but we can not specify exactly what was modified and the only certainty is when the audited chat is correct and complete. This is in line with what is expected of the system and its specification.

## 9 CONCLUSION

In this work, we presented the importance of instant messaging apps and show the security gap present in these systems, in particular, the difficulty to audit conversations in these apps. The proposed architecture aims to mitigate vulnerabilities of instant messaging apps, developing a communication structure that is secure and allows for an audit of conversations carried out on IM systems. For this, we have used a DHT network to allow users' connections in this distributed system, a PGP web of trust to authenticate users, and a hash chain to record the messages exchanges. With this, we expected to add integrity, authenticity, resilience, and non-repudiation characteristics to the platform. We also cared about privacy including a selective disclosure feature that allowed a partial and reliable audit.

As described in this work, each mechanism presents an independent security feature (integrity, privacy, confidentiality, availability, authenticity). Other IM apps available in the market present some of the mechanisms used as a solution in this work, so the main contribution that we proposed is the use of hash chain structure to generate a chat's log that can be audited. One point raised during this work was the use of blockchain technology, due to this popularity and the similarity between the bitcoin ledger and the chat register that we aimed for. However, analyzing the requisites and scope we concluded that the use of blockchain technology was excessive, because the consensus algorithms used in the blockchain would derail the communication in the IM app. This level of consensus is not necessary, so the hash chain structure proposed is enough.

To analyze the proposed architecture, first, an Android app was developed as a proof of concept. Due to difficulties in developing a completed and functional app with all proposed features, the Android PoC focused on the efficiency test. We analyzed what impact cryptographic functions would have on communication. According to our tests, the modifications made to the Android PoC messaging app did not impact the system performance significantly, showing that our solution is efficient enough for message exchange purposes. Second, a Python web chat was developed as a proof of concept, focusing on the

generation, construction, and validation of the conversation with hash chain. The tests in Python chat PoC were also satisfying, we could confirm that it is possible to perform a reliable audit in the hash chain as we intended.

Following, we list the related articles published and describe the future works that can be continued from this master’s research.

## 9.1 Publications

Resulting from the research carried out during this work, we produced the following publications:

- Conference Paper: KOMO, A. E.; ARAKAKI, B. O.; SIMPLICIO JR., M. A.; LEVY, M. R. Aplicativo de troca de mensagens instantâneas utilizando comunicação P2P. In: Anais Estendidos do XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. Porto Alegre, RS, Brasil: SBC, 2018. p.65–72. Available in: <[https://sol.sbc.org.br/index.php/sbseg\\_estendido/article/view/4143](https://sol.sbc.org.br/index.php/sbseg_estendido/article/view/4143)>
- Conference Paper: KOMO, A. E.; SIMPLICIO JR., M. A. Solução para habilitar conversas integras e auditáveis em aplicativos de troca de mensagens instantâneas. In: Anais do XIX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. Porto Alegre, RS, Brasil: SBC, 2019. Available in: <<https://sbseg2019.ime.usp.br/anais/196912.pdf>>

The paper “Solução para habilitar conversas integras e auditáveis em aplicativos de troca de mensagens instantâneas” mentioned above was awarded as the best complete article at the XIII Workshop de Trabalhos de Iniciação Científica e de Graduação (WTICG) of the XIX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg).

## 9.2 Future Works

In order to disseminate this work, we plan to submit it to a conference or journal, as the papers already submitted did not present the selective disclosure and agree-to-disagree features. The results obtained in this work were satisfactory and they can be incorporated into an IM app open source to generate a marketable app that is competitive

with popular apps available on the market. Other future work would be to develop an independent system just for audits. If many IM apps use the same logic to generate the hash chain, with one single system will be possible to check and audit any of these chats. Moreover, to generate a marketable app, it will be necessary to consider the items excluded from the scope in Section 3.1, like sending messages containing attached files, photos, videos, and group chats.

## REFERENCES

- ABU-SALMA, R.; SASSE, M. A.; BONNEAU, J.; DANILOVA, A.; NAIKSHINA, A.; SMITH, M. Obstacles to the adoption of secure communication tools. In: *2017 IEEE Symposium on Security and Privacy (SP)*. San Jose, CA: IEEE and IACR, 2017. p. 137–153. ISSN 2375-1207.
- ARAKAKI, B. O.; LEVY, M. R. *Aplicação Móvel de troca de mensagens utilizando comunicação P2P*. 2017. Monografia (Trabalho de Conclusão de Curso com ênfase em Engenharia de Computação) - Escola Politécnica da Universidade de São Paulo.
- BARRETO, I.; LIMA, M. Marco Civil da Internet: Análise das Decisões Judiciais que Suspenderam o Aplicativo WhatsApp no Brasil – 2015-16. *Rev. de Direito, Governança e Novas Tecnologias*, v. 2, n. 2, 2016.
- BERNSTEIN, D. J. Cryptography in NaCl. 01 2009. Available at: <<https://cr.yp.to/highspeed/naclcrypto-20090310.pdf>>. Accessed Aug. 2021.
- BOCEK, T. *TomP2P: A P2P-based high performance key-value pair storage library*. 2004. Available at: <<https://tomp2p.net/>>. Accessed Feb. 2019.
- BRASIL. *Mensagens de Whatsapp é meio de prova judicial*. 2017. Available at: <<https://acintiazc.jusbrasil.com.br/noticias/412703162/mensagem-de-whatsapp-e-meio-de-prova-judicial>>. Accessed: Apr. 2018.
- \_\_\_\_\_. *LEI Nº 13.709, DE 14 DE AGOSTO DE 2018. Lei Geral de Proteção de Dados Pessoais (LGPD)*. 2018. Available at: <[http://www.planalto.gov.br/ccivil\\_03/\\_Ato2015-2018/2018/Lei/L13709.htm](http://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2018/Lei/L13709.htm)>. Accessed Apr. 2021.
- BROWN, L.; STALLINGS, W. *Segurança de Computadores: Princípios e Práticas*. 2. ed. Rio de Janeiro: Elsevier Editora Ltda., 2015. ISBN 9788535264500.
- CALLAS, J.; DONNERHACKE, L.; FINNEY, H.; SHAW, D.; THAYER, R. *OpenPGP Message Format*. RFC Editor, 2007. 1-90 p. Available at: <<https://www.rfc-editor.org/rfc/rfc4880.txt>>. Accessed Apr. 2021.
- Certicom Research. *SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0*. 2000. Available at: <<http://www.secg.org/SEC2-Ver-1.0.pdf>>. Accessed July 2021.
- CHÁVEZ, A. G. M.; CORTÉS, E. P.; GUERRERO, M. L. A performance comparison of Chord and Kademlia DHTs in high churn scenarios. *Peer-to-Peer Networking and Applications*, v. 8, n. 5, p. 807–821, Sep 2015. ISSN 1936-6450. Available at: <<https://doi.org/10.1007/s12083-014-0294-y>>. Accessed Sept. 2019.
- CICHINI, P. *BMW Group Brasil estreia canal digital no WhatsApp*. 2021. Available at: <<https://www.press.bmwgroup.com/brazil/article/detail/T0323929PT/bmw-group-brasil-estrela-canal-digital-no-whatsapp>>. Accessed Dec. 2021.

COHN-GORDON, K.; CREMERS, C.; DOWLING, B.; GARRATT, L.; STEBILA, D. A formal security analysis of the signal messaging protocol. In: *IEEE European Symposium on Security and Privacy (EuroS&P)*. Paris, France: IEEE, 2017. p. 451–466.

DIFFIE, W.; HELLMAN, M. E. Multiuser cryptographic techniques. In: *Proceedings of the June 7-10, 1976, National Computer Conference and Exposition*. New York, NY, USA: Association for Computing Machinery, 1976. (AFIPS '76), p. 109–112. ISBN 9781450379175. Available at: <<https://doi.org/10.1145/1499799.1499815>>. Accessed Jan. 2022.

DUROV, P. *Durov's Channel in Telegram*. 2021. Available at: <<https://t.me/durov/147>>. Accessed Nov. 2021.

ELAHI, G.; YU, E. A Goal Oriented Approach for Modeling and Analyzing Security Trade-Offs. In: PARENT, C.; SCHEWE, K.-D.; STOREY, V. C.; THALHEIM, B. (Ed.). *Conceptual Modeling - ER 2007*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 375–390. ISBN 978-3-540-75563-0.

FACEBOOK. *1.3 Billion people on messenger*. 2017. Available at: <<https://www.facebook.com/messenger/posts/more-than-13-billion-people-around-the-world-now-use-messenger-every-month-weve-1530169047102770/>>. Accessed Nov. 2021.

FUNK, M.; CUNNINGHAM, C.; KANVER, D.; SAIKALIS, C.; PANSARE, R. Usable and acceptable response delays of conversational agents in automotive user interfaces. In: *12th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. New York, NY, USA: Association for Computing Machinery, 2020. (AutomotiveUI '20), p. 262–269. ISBN 9781450380652. Available at: <<https://doi.org/10.1145/3409120.3410651>>. Accessed June 2022.

GARTNER. *Gartner Says Worldwide Sales of Smartphones Recorded First Ever Decline During the Fourth Quarter of 2017*. 2018. Available at: <<https://www.gartner.com/newsroom/id/3859963>>. Accessed July 2018.

GOMAA, H. Software modeling and design: Uml, use cases, patterns, and software architectures. *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*, p. 1–550, 01 2011.

GULTSCH, D. *Conversations*. 2014. Available at: <<https://conversations.im/>>. Accessed Aug. 2021.

HU, Y.-C.; JAKOBSSON, M.; PERRIG, A. Efficient constructions for one-way hash chains. In: *Proc. of the 3rd Int. Conf. on Applied Cryptography and Network Security (ACNS'05)*. Berlin, Heidelberg: Springer-Verlag, 2005. p. 423–441. ISBN 3-540-26223-7, 978-3-540-26223-7.

JOHNSON, D.; MENEZES, A.; VANSTONE, S. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, v. 1, n. 1, p. 36–63, Aug 2001. ISSN 1615-5262. Available at: <<https://doi.org/10.1007/s102070100002>>. Accessed Jan. 2022.

- KEATES, S. Measuring acceptable input: What is “good enough”? In: . [S.l.]: Universal Access in the Information Society volume 16, 2016. p. 713–723. Available at: <<https://doi.org/10.1007/s10209-016-0498-4>>. Accessed June 2022.
- KOMO, A. E.; ARAKAKI, B. O.; SIMPLICIO JR., M. A.; LEVY, M. R. Aplicativo de troca de mensagens instantâneas utilizando comunicação P2P. In: *Anais Estendidos do XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*. Porto Alegre, RS, Brasil: SBC, 2018. p. 65–72. Available at: <[https://sol.sbc.org.br/index.php/sbseg\\_estendido/article/view/4143](https://sol.sbc.org.br/index.php/sbseg_estendido/article/view/4143)>. Accessed Jan. 2019.
- KOMO, A. E.; SIMPLICIO JR., M. A. Solução para habilitar conversas integras e auditáveis em aplicativos de troca de mensagens instantâneas. In: *Anais do XIX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*. Porto Alegre, RS, Brasil: SBC, 2019. Available at: <<https://sbseg2019.ime.usp.br/anais/196912.pdf>>. Accessed Jan. 2020.
- KUROSE, J.; ROSS, K. *Redes de computadores e a internet: uma abordagem top-down*. 6. ed. São Paulo: Pearson Education do Brasil, 2013. ISBN 9788581436777.
- LAMACCHIA, B.; LAUTER, K.; MITYAGIN, A. Stronger security of authenticated key exchange. In: SUSILO, W.; LIU, J. K.; MU, Y. (Ed.). *Provable Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 1–16. ISBN 978-3-540-75670-5.
- MARLINSPIKE, M.; PERRIN, T. *The double ratchet algorithm*. 2016. Available at: <<https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf>>. Accessed Aug. 2021.
- MAYMOUNKOV, P.; MAZIÈRES, D. Kademia: A peer-to-peer information system based on the xor metric. In: DRUSCHEL, P.; KAASHOEK, F.; ROWSTRON, A. (Ed.). *Peer-to-Peer Systems. IPTPS 2002*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. p. 53–65. ISBN 978-3-540-45748-0.
- MAZIÈRES, D. *Self-certifying File System*. Tese (Doutorado) — MIT, 2000.
- MCGREW, D. A.; VIEGA, J. *The Galois/counter mode of operation (GCM)*. 2004.
- MEREDITH, S. *Facebook-Cambridge Analytica: A timeline of the data hijacking scandal*. Nova Jersey, EUA: CNBC, 2018. Available at: <<https://www.cnbc.com/2018/04/10/facebook-cambridge-analytica-a-timeline-of-the-data-hijacking-scandal.html>>. Accessed Aug. 2018.
- MEYER, U.; WETZEL, S. A Man-in-the-middle Attack on UMTS. In: *Proceedings of the 3rd ACM Workshop on Wireless Security*. New York, NY, USA: ACM, 2004. (WiSe '04), p. 90–97. ISBN 1-58113-925-X. Available at: <<http://doi.acm.org/10.1145/1023646.1023662>>. Accessed May 2019.
- NAKAMOTO, S. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008.
- NAYEBI, F.; DESHARNAIS, J.-M.; ABRAN, A. The state of the art of mobile application usability evaluation. In: *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. Quebec, Canada: IEEE, 2012. p. 1–4.



- NELSON, B.; ASKAROV, A. With a little help from my friends: Transport deniability for instant messaging. *CoRR*, abs/2202.02043, 2022. Available at: <<https://arxiv.org/abs/2202.02043>>. Accessed May 2021.
- NIST. *FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Gaithersburg, MD, 2015. Doi:10.6028/NIST.FIPS.202.
- RESCORLA, E. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC Editor, 2018. 1-160 p. Available at: <<https://www.rfc-editor.org/rfc/rfc8446.txt>>. Accessed Apr. 2021.
- ROGERS, M.; SAITTA, E.; GROTE, T.; DEHM, J.; ERLINGSSON, E.; TYERS, B.; GRIGG, J. *Briar*. 2018. Available at: <<https://briarproject.org/>>. Accessed Feb. 2019.
- RÖSLER, P.; MAINKA, C.; SCHWENK, J. More is less: On the end-to-end security of group chats in Signal, WhatsApp, and Threema. In: *IEEE European Symposium on Security and Privacy (Euro S&P)*. London, UK: 2018 IEEE European Symposium on Security and Privacy (Euro S&P), 2018. p. 415–429.
- SAINT-ANDRE, P. *Extensible Messaging and Presence Protocol (XMPP): Core*. RFC Editor, 2011. 1-211 p. Available at: <<https://www.rfc-editor.org/rfc/rfc6120.txt>>. Accessed Apr. 2021.
- SCHLIEP, M.; HOPPER, N. *End-to-End Secure Mobile Group Messaging with Conversation Integrity and Deniability*. 2018. Cryptology ePrint Archive, Report 2018/1097. Available at: <<https://eprint.iacr.org/2018/1097>>. Accessed Dec. 2019.
- SCHLIEP, M.; KARINIEMI, I.; HOPPER, N. Is Bob sending mixed signals? In: *Proc. of the 2017 on Workshop on Privacy in the Electronic Society*. New York, NY, USA: ACM, 2017. (WPES '17), p. 31–40. ISBN 978-1-4503-5175-1.
- SCHRITTWIESER, S.; FRÜHWIRT, P.; KIESEBERG, P.; LEITHNER, M.; MULAZZANI, M.; HUBER, M.; WEIPPL, E. R. Guess who's texting you? evaluating the security of smartphone messaging applications. In: *in Proceedings of the 19th Annual Symposium on Network and Distributed System Security (NDSS)*. San Diego, California: Internet Society, 2012.
- SIMPLICIO, M.; OLIVEIRA, B.; MARGI, C.; BARRETO, P.; CARVALHO, T.; NÄSLUND, M. Survey and comparison of message authentication solutions on wireless sensor networks. *Ad Hoc Networks*, Elsevier, Amsterdam, The Netherlands, v. 11, n. 3, p. 1221–1236, maio 2013. ISSN 1570-8705.
- STALLINGS, W. *Criptografia e Segurança de Redes*. 6. ed. São Paulo: Pearson Education do Brasil, 2014. ISBN 9788543005898.
- STOICA, I.; MORRIS, R.; LIBEN-NOWELL, D.; KARGER, D. R.; KAASHOEK, M. F.; DABEK, F.; BALAKRISHNAN, H. Chord: a scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, v. 11, n. 1, p. 17–32, Feb 2003. ISSN 1063-6692.
- STRAUB, A.; GULTSCH, D.; HENKES, T.; HERBERTH, K.; SCHAUB, P.; WISFELD, M. XEP, *OMEMO Encryption*. 2015. Available at: <<https://xmpp.org/extensions/xep-0384.html>>. Accessed Aug. 2021.

TELEGRAM. *Telegram FAQ [online]*. 2018. Available at: <<https://telegram.org/faq>>. Accessed Dec. 2019.

\_\_\_\_\_. *MTPProto Mobile Protocol [online]*. 2019. Available at: <<https://core.telegram.org/mtproto>>. Accessed Aug. 2021.

THE EUROPEAN PARLIAMENT; THE COUNCIL OF THE EUROPEAN UNION. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)*. 2016. Available at: <<http://data.europa.eu/eli/reg/2016/679/oj>>. Accessed Apr. 2021.

THREEMA. *Cryptography Whitepaper*. 2021. Available at: <[https://threema.ch/press-files/2\\_documentation/cryptography\\_whitepaper.pdf](https://threema.ch/press-files/2_documentation/cryptography_whitepaper.pdf)>. Accessed Aug. 2021.

TYLEY, R. *Spongy Castle - a repackage of Bouncy Castle for Android*. 2014. Available at: <<https://rtyley.github.io/spongycastle/>>. Accessed Apr. 2021.

WEISER, M. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 3, n. 3, p. 3–11, jul. 1999. ISSN 1559-1662. Available at: <<http://doi.acm.org/10.1145/329124.329126>>. Accessed Mar. 2019.

WHATSAPP. *WhatsApp Encryption Overview*. 2016. Available at: <<https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf>>. Accessed May 2018.

\_\_\_\_\_. *Two Billion Users – Connecting the World Privately*. 2020. Available at: <<https://blog.whatsapp.com/two-billion-users-connecting-the-world-privately>>. Accessed Apr. 2021.

\_\_\_\_\_. *Send and receive money right where you chat*. 2021. Available at: <<https://www.whatsapp.com/payments/>>. Accessed Apr. 2021.

ZHANG, H.; WEN, Y.; XIE, H.; YU, N. *A Survey on Distributed Hash Table (DHT): Theory, Platforms, and Applications*. 2013.

ZIMMERMANN, P. *Pretty Good Privacy: Public Key Encryption for the Masses*. New York, NY: Springer New York, 1995. 93–107 p. Available at: <[https://doi.org/10.1007/978-1-4612-2524-9\\_2](https://doi.org/10.1007/978-1-4612-2524-9_2)>. Accessed Apr. 2021. ISBN 978-1-4612-2524-9.

## APPENDIX A – BENCHMARK TABLES

In this appendix, the information from the Chapter 8's tables has been merged into one table in order to facilitate the comparative analysis between the devices.

Table 23: SHA3-256 benchmark

Message's size (#characters)	Samsung Galaxy J2 Prime			How HT-705 tablet			Xiaomi Redmi Note 7			Samsung Galaxy A22		
	$\bar{t}$	$\sigma$	$\tilde{t}$	$\bar{t}$	$\sigma$	$\tilde{t}$	$\bar{t}$	$\sigma$	$\tilde{t}$	$\bar{t}$	$\sigma$	$\tilde{t}$
5	0.310	0.504	0.000	1.018	1.554	1.000	0.068	0.260	0.000	0.060	0.246	0.000
50	0.254	0.435	0.000	0.298	0.503	0.000	0.020	0.140	0.000	0.024	0.153	0.000
100	0.272	0.445	0.000	0.238	0.426	0.000	0.024	0.153	0.000	0.032	0.176	0.000
200	0.470	0.503	0.000	0.340	0.474	0.000	0.042	0.201	0.000	0.024	0.153	0.000
500	0.904	0.308	1.000	0.742	1.586	1.000	0.058	0.234	0.000	0.042	0.201	0.000
1000	1.822	0.463	2.000	1.066	2.095	1.000	0.088	0.283	0.000	0.072	0.258	0.000

Source: Authors.

Table 24: PGP encryption benchmark

Message's size (#characters)	Samsung Galaxy J2 Prime			How HT-705 tablet			Xiaomi Redmi Note 7			Samsung Galaxy A22		
	$\bar{t}$	$\sigma$	$\tilde{t}$	$\bar{t}$	$\sigma$	$\tilde{t}$	$\bar{t}$	$\sigma$	$\tilde{t}$	$\bar{t}$	$\sigma$	$\tilde{t}$
5	4.198	0.956	4.000	10.706	4.391	9.000	3.926	0.986	4.000	2.584	0.927	2.000
50	4.240	0.531	4.000	8.574	1.117	8.000	5.326	1.563	6.000	2.870	1.202	2.000
100	4.258	0.707	4.000	9.198	3.074	9.000	6.056	0.530	6.000	4.072	0.774	4.000
200	4.386	0.530	4.000	8.786	0.976	9.000	5.952	0.523	6.000	4.036	0.771	4.000
500	4.740	0.544	5.000	9.570	2.259	9.000	6.044	0.535	6.000	4.100	0.833	4.000
1000	5.292	0.561	5.000	9.866	0.963	10.000	6.156	0.521	6.000	4.116	0.794	4.000

Source: Authors.

Table 25: PGP decryption benchmark

Message's size (#characters)	Samsung Galaxy J2 Prime			How HT-705 tablet			Xiaomi Redmi Note 7			Samsung Galaxy A22		
	$\bar{t}$	$\sigma$	$\tilde{t}$	$\bar{t}$	$\sigma$	$\tilde{t}$	$\bar{t}$	$\sigma$	$\tilde{t}$	$\bar{t}$	$\sigma$	$\tilde{t}$
5	89.952	4.850	88.000	66.982	9.949	65.000	13.160	1.796	13.000	13.140	2.317	13.000
50	89.710	3.871	88.000	60.966	3.807	61.000	19.284	5.247	22.000	15.808	5.691	12.500
100	90.354	4.637	88.000	63.948	12.076	62.000	22.726	1.229	22.000	24.448	1.571	24.000
200	90.022	3.898	88.000	61.518	3.372	61.000	22.714	1.210	22.000	24.392	1.626	24.000
500	90.980	4.306	89.000	64.140	9.775	62.000	22.894	1.282	22.000	24.464	1.695	24.000
1000	92.046	4.367	90.000	63.142	3.726	63.000	23.214	1.452	23.000	24.612	1.557	24.000

Source: Authors.