

UNIVERSIDADE FEDERAL DO PARANÁ

ELTON EIJI SASAKI

USE OF BLOCKCHAIN TIMESTAMPING AND DIGITAL CERTIFICATES BASED
ON ICP-BRASIL STANDARDS TO PROVIDE AUTHENTICITY OF DOCUMENTS

CURITIBA

2020

ELTON EIJI SASAKI

USE OF BLOCKCHAIN TIMESTAMPING AND DIGITAL CERTIFICATES BASED
ON ICP-BRASIL STANDARDS TO PROVIDE AUTHENTICITY OF DOCUMENTS

Dissertação apresentada ao curso de Pós-Graduação em Gestão da Informação, Setor de Ciências Sociais Aplicadas, Universidade Federal do Paraná, como requisito parcial à obtenção do título de Mestre em Gestão da Informação.

Orientador: Prof. Dr. Egon Walter Wildauer

CURITIBA

2020

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DE CIÊNCIAS SOCIAIS
APLICADAS – SIBI/UFPR COM DADOS FORNECIDOS PELO(A) AUTOR(A)
Bibliotecário: Eduardo Silveira – CRB 9/1921

Sasaki, Elton Eiji
Use of blockchain timestamping and digital certificates based on
ICP-Brasil standards to provide authenticity of documents / Elton Eiji
Sasaki.- 2020.
147 p.

Dissertação (Mestrado) - Universidade Federal do Paraná. Programa
de Pós-Graduação em Gestão da Informação, do Setor de Ciências
Sociais Aplicadas.

Orientador: Egon Walter Wildauer.

Defesa: Curitiba, 2020.

1. Blockchain. 2. Certificado digital. 3. Assinaturas digitais.
I. Universidade Federal do Paraná. Setor de Ciências Sociais Aplicadas.
Programa de Pós-Graduação em Gestão da Informação. II. Wildauer,
Egon Walter. III. Título.

CDD 658.47



MINISTÉRIO DA EDUCAÇÃO
SETOR DE CIÊNCIAS SOCIAIS E APLICADAS
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO GESTÃO DA
INFORMAÇÃO - 40001016058P1

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em GESTÃO DA INFORMAÇÃO da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **ELTON EIJI SASAKI** intitulada: **USE OF BLOCKCHAIN TIMESTAMPING AND DIGITAL CERTIFICATES BASED ON ICP-BRASIL STANDARDS TO PROVIDE AUTHENTICITY OF DOCUMENTS**, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua **APROVAÇÃO** no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 30 de Março de 2020.

Assinatura Eletrônica
23/03/2020 13:26:05.0
EGON WALTER WILDAUER
Presidente da Banca Examinadora

Assinatura Eletrônica
31/03/2020 22:38:20.0
ANA CRISTINA BARREIRAS KOICHEM VENDRAMIN
Avaliador Externo (UNIVERSIDADE TECNOLÓGICA FEDERAL DO
PARANÁ)

Assinatura Eletrônica
31/03/2020 10:20:06.0
TAIANE RITTA COELHO
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
23/04/2020 15:45:27.0
CARLOS HENRIQUE KURETZKI
Avaliador Externo (UNIVERSIDADE POSITIVO)

AGRADECIMENTOS

Agradeço ao meu orientador Professor Egon Walter Wildauer pela mentoria e direcionamento na realização deste trabalho.

Agradeço aos Professores, Professor Egon Walter Wildauer, Professor José Simão de Paula Pinto e Professora Helena de Fátima Nunes Silva, pelas aulas ministradas durante o curso de mestrado que me fortaleceram academicamente e profissionalmente.

Agradeço aos membros das bancas de qualificação e defesa, Professor Egon Walter Wildauer, Professor José Simão de Paula Pinto, Professora Ana Cristina Barreiras Kochem Vendramin, Professora Taiane Ritta Coelho e Professor Carlos Henrique Kuretzki, pelas considerações e correções apontadas para o desenvolvimento deste trabalho.

ACKNOWLEDGMENTS

I would like to thank my supervisor Professor Egon Walter Wildauer for mentoring and directing this work.

I would like to thank Professors, Professor Egon Walter Wildauer, Professor José Simão de Paula Pinto and Professor Helena de Fátima Nunes Silva, for the classes given during the master's course that strengthened me academically and professionally.

I would like to thank the members of the qualification and defense boards, Professor Egon Walter Wildauer, Professor José Simão de Paula Pinto, Professor Ana Cristina Barreiras Kochem Vendramin, Professor Taiane Ritta Coelho and Professor Carlos Henrique Kuretzki, for the considerations and corrections pointed out for the development of this work.

RESUMO

Este projeto de pesquisa tem como objetivo implementar a integração de carimbo de tempo de blockchain e assinaturas digitais baseadas em certificados digitais ICP-Brasil para fornecer um serviço ubíquo de autenticação digital de documentos por meio de um protótipo de aplicativo para dispositivos Android. Este projeto de pesquisa tem três objetivos específicos: a) identificar como a integração de carimbo de tempo de blockchain e assinaturas digitais baseadas em certificados digitais ICP-Brasil garante a segurança da informação; b) identificar requisitos de sistema para desenvolver o protótipo do aplicativo para dispositivos Android que integre carimbo de tempo de blockchain e assinaturas digitais baseadas em certificados digitais ICP-Brasil; e c) desenvolver o protótipo do aplicativo para dispositivos Android seguindo os requisitos de sistema identificados no item b. A integração das tecnologias blockchain e certificados digitais ICP-Brasil (Infraestrutura de Chaves Públicas Brasileira) simplifica o desenvolvimento de um serviço de autenticação digital de documentos. A ICP-Brasil fornece um serviço terceirizado de KYC (Know Your Customer - Conheça o seu cliente), verificando a identidade e o comprovante de residência do usuário, antes de emitir um certificado digital para o usuário. E o blockchain fornece um serviço terceirizado de registro de carimbos de tempo à prova de violações.

Palavras-chave: Blockchain. Infraestrutura de Chaves Públicas. ICP-Brasil. Certificados Digitais. Assinaturas Digitais. Autenticação Digital de Documentos.

ABSTRACT

This paper aims to integrate blockchain timestamping and digital signatures based on ICP-Brasil digital certificates to provide a ubiquitous digital document authentication service through an application prototype for mobile Android devices. This paper has three specific objectives: a) identify how integration of blockchain timestamping and digital signatures based on ICP-Brasil digital certificates assures information security; b) identify system requirements to develop the application prototype for mobile Android devices that integrates blockchain timestamping and digital signatures based on ICP-Brasil; and c) follow system requirements identified in item b in order to develop the application prototype for mobile Android devices. Integration of blockchain and ICP-Brasil (Infraestrutura de Chaves Públicas Brasileira - Brazilian Public Key Infrastructure) simplifies the development of a digital document authentication service. ICP-Brasil provides an outsourced KYC (Know Your Customer) service by verifying through a Registration Authority the user's identity and proof of address, prior to issuing a digital certificate to the user. And blockchain provides an outsourced tamper-proofing timestamp data.

Keywords: Blockchain. Public Key Infrastructure. ICP-Brasil. Digital Certificates. Digital Signatures. Digital Document Authentication.

LIST OF FIGURES

FIGURE 1 – CRYPTOCURRENCY EXCHANGES BY COUNTRIES	23
FIGURE 2 – COUNTRIES WITH MOST CRYPTOCURRENCY USERS	24
FIGURE 3 – DATA OF INTERNET USE IN BRAZIL	24
FIGURE 4 – FLOWCHART OF THESIS STAGES	27
FIGURE 5 – CHAIN OF DIGITAL SIGNATURES.....	29
FIGURE 6 – BLOCKS LINKED IN A CHAIN TO THE PREVIOUS BLOCK HEADER HASH	32
FIGURE 7 – BITCOIN HIGHEST AVERAGE NUMBER OF TRANSACTIONS PER BLOCK.....	34
FIGURE 8 – BITCOIN AVERAGE NUMBER OF TRANSACTIONS PER BLOCK ON MARCH 11, 2020	35
FIGURE 9 – CALCULATING NODES IN A MERKLE TREE	38
FIGURE 10 – BRAZILIAN PUBLIC KEY INFRASTRUCTURE.....	41
FIGURE 11 – METHODOLOGY MAP	50
FIGURE 12 – APPLICATION PROTOTYPE USE CASE DIAGRAM	53
FIGURE 13 – PROCESS FLOWCHART OF APPLICATION PROTOTYPE	54
FIGURE 14 – APPLICATION PROTOTYPE ENTITY-RELATIONSHIP DIAGRAM ..	55
FIGURE 15 – IMPORT DIGITAL CERTIFICATE USE CASE DIAGRAM	57
FIGURE 16 – ANDROID APP IMPORT DIGITAL CERTIFICATE CLASS DIAGRAM	58
FIGURE 17 – IMPORT DIGITAL CERTIFICATE SEQUENCE DIAGRAM.....	60
FIGURE 18 – FUNCTIONALITIES MENU ON ANDROID APP.....	61
FIGURE 19 – “SIGNATURES” FUNCTIONALITY ON ANDROID APP	62
FIGURE 20 – FILE MANAGER TO SELECT CERTIFICATE	63
FIGURE 21 – DIGITAL CERTIFICATE PASSWORD VALIDATOR	64
FIGURE 22 – DOCUMENT SIGNING USE CASE DIAGRAM	65
FIGURE 23 - ANDROID APP DOCUMENT SIGNING CLASS DIAGRAM.....	66
FIGURE 24 – DOCUMENT SIGNING SEQUENCE DIAGRAM	68
FIGURE 25 – FUNCTIONALITIES MENU ON ANDROID APP.....	69
FIGURE 26 – “SIGNATURES” FUNCTIONALITY ON ANDROID APP	70
FIGURE 27 – FILE MANAGER TO SELECT SIGNED DOCUMENT	71
FIGURE 28 – DIGITAL CERTIFICATE PASSWORD VALIDATOR	72

FIGURE 29 – DOCUMENT AUTHENTICATION USE CASE DIAGRAM	73
FIGURE 30 – ANDROID APP DOCUMENT AUTHENTICATION CLASS DIAGRAM	75
FIGURE 31 – CLASS DIAGRAM OF DOCUMENT AUTHENTICATION	77
FIGURE 32 – DOCUMENT AUTHENTICATION SEQUENCE DIAGRAM	79
FIGURE 33 – FUNCTIONALITIES MENU ON ANDROID APP.....	80
FIGURE 34 – “NOTARIZATIONS” FUNCTIONALITY ON ANDROID APP	81
FIGURE 35 – FILE MANAGER TO SELECT SIGNED DOCUMENT FOR NOTARIZATION	82
FIGURE 36 – BITCOIN WALLET	83
FIGURE 37 – SUCCESS AUTHENTICATION MESSAGE	84
FIGURE 38 – VERIFICATION OF DOCUMENT AUTHENTICATION USE CASE DIAGRAM	85
FIGURE 39 – ANDROID APP VERIFICATION OF DOCUMENT AUTHENTICATION CLASS DIAGRAM.....	87
FIGURE 40 –VERIFICATION OF DOCUMENT AUTHENTICATION CLASS DIAGRAM	88
FIGURE 41 – DOCUMENT AUTHENTICATION VERIFICATION SEQUENCE DIAGRAM	90
FIGURE 42 – FUNCTIONALITIES MENU ON ANDROID APP.....	91
FIGURE 43 – TRANSACTION ID VERIFICATION.....	92
FIGURE 44 – FILE MANAGER TO SELECT SIGNED DOCUMENT FOR VERIFICATION OF AUTHENTICATION.....	93
FIGURE 45 – VERIFIED DOCUMENT INFORMATION.....	94
FIGURE 46 – BITCOIN - ETHEREUM – ETHEREUM CLASSIC AVERAGE HASHRATE PER DAY.....	107
FIGURE 47 – BITCOIN - ETHEREUM – ETHEREUM CLASSIC AVERAGE BLOCK TIMESTAMP INTERVALS IN MINUTES.....	108
Figure 48 – BITCOIN – ETHEREUM - ETHEREUM CLASSIC AVERAGE TRANSACTION FEE PER DAY A	109
Figure 49 – BITCOIN – ETHEREUM - ETHEREUM CLASSIC AVERAGE TRANSACTION FEE PER DAY B	109

LIST OF TABLES

TABLE 1 – COMPARISON BETWEEN TOTAL ISSUED DIGITAL CERTIFICATES AND TOTAL INCOME TAX COLLECTION	22
TABLE 2 – PROPERTIES OF SHA HASH ALGORITHMS	37
TABLE 3 – RELATED WORKS FEATURES	49
TABLE 4 – COST OF ISSUING DIGITAL CERTIFICATES ON CERTISIGN AS OF MAY 23, 2020	105

LIST OF CHARTS

CHART 1 – DIGITAL CERTIFICATES ISSUED YEAR BY YEAR.....	19
CHART 2 – TOTAL LEGAL ENTITY INCOME TAX COLLECTION – IPCA INDEX ..	21

LIST OF CODE EXCERPTS

CODE EXCERPT 1 - IMPORT DIGITAL CERTIFICATE OPERATION.....	96
CODE EXCERPT 2 – DOCUMENT SIGNING OPERATION	98
CODE EXCERPT 3 – MAINNET AND TESTNET BLOCKCHAINS.....	100
CODE EXCERPT 4 – PAYMENTS LISTENER.....	101
CODE EXCERPT 5 – TIMESTAMP DATA.....	102
CODE EXCERPT 6 – VERIFICATION OF DOCUMENT AUTHENTICATION	103

LIST OF ABBREVIATIONS AND ACRONYMS

API	-	Application Programming Interface
CA	-	Certification Authority
CRL	-	Certificate Revocation List
ER	-	Entity-Relationship
IaaS	-	Infrastructure as a Service
FIPS	-	Federal Information Processing Standards
GUI	-	Graphical User Interface
ICP-BRASIL	-	Infraestrutura de Chaves Públicas Brasileira (Brazilian Public Key Infrastructure)
ITI	-	Instituto Nacional de Tecnologia da Informação
JSON	-	JavaScript Object Notation
KYC	-	Know Your Customer
MEI	-	Micro Empreendedor Individual
MTE	-	Ministério do Trabalho e Emprego
NIST	-	National Institute of Standards and Technology
NGO	-	Non-Governmental Organization
PKI	-	Public Key Infrastructure
RA	-	Registration Authority
RL	-	Revocation List
TA	-	Timestamp Authority
UML	-	Unified Modeling Language
VM	-	Virtual Machine

SUMMARY

1 INTRODUCTION	17
1.1 CONTEXT	18
1.1.1 PESTLE.....	18
1.2 JUSTIFICATION.....	25
1.3 OBJECTIVES	26
1.3.1 GENERAL OBJECTIVE	26
1.3.2 SPECIFIC OBJECTIVES.....	26
1.4 THESIS STRUCTURE	26
2 LITERATURE REVIEW	28
2.1 BITCOIN CRYPTOCURRENCY.....	28
2.1.1 BLOCKCHAIN	28
2.1.2 TRANSACTIONS	29
2.1.3 DOUBLE-SPENDING.....	30
2.1.4 PROOF-OF-WORK	30
2.1.5 BLOCK HEADER	33
2.1.6 TIMESTAMPING IN THE BLOCKCHAIN	33
2.2 SHA 256 ALGORITHM.....	36
2.3 PRIVATE-KEY OR SYMMETRIC-KEY ENCRYPTION	38
2.4 PUBLIC-KEY OR ASYMMETRIC-KEY ENCRYPTION	39
2.5 PUBLIC-KEY INFRASTRUCTURE	40
2.6 BRAZILIAN PUBLIC KEY INFRASTRUCTURE (ICP-BRASIL).....	41
2.7 INFORMATION SECURITY	43
2.8 SMART CONTRACTS.....	46
2.9 RELATED WORKS	47
3 METHODOLOGY	50
3.1 RESEARCH CHARACTERIZATION	51
3.2 DELIMITATION	51
3.3 REQUIREMENTS OF APPLICATION PROTOTYPE.....	52
3.3.1 USE CASE DIAGRAMS	52
3.3.2 PROCESS FLOWCHART	53
3.3.3 DATABASE	55
3.3.4 IMPORT DIGITAL CERTIFICATE	57

3.3.4.1 USE CASE DIAGRAM OF IMPORT DIGITAL CERTIFICATE.....	57
3.3.4.2 CLASS DIAGRAM OF IMPORT DIGITAL CERTIFICATE	57
3.3.4.3 SEQUENCE DIAGRAM OF IMPORT DIGITAL CERTIFICATE.....	59
3.3.5 DOCUMENT SIGNING.....	65
3.3.5.1 USE CASE DIAGRAM OF DOCUMENT SIGNING	65
3.3.5.2 CLASS DIAGRAM OF DOCUMENT SIGNING	65
3.3.5.3 SEQUENCE DIAGRAM OF DOCUMENT SIGNING	67
3.3.6 DOCUMENT AUTHENTICATION	73
3.3.6.1 USE CASE DIAGRAM OF DOCUMENT AUTHENTICATION.....	73
3.3.6.2 CLASS DIAGRAMS OF DOCUMENT AUTHENTICATION.....	73
3.3.6.3 SEQUENCE DIAGRAM OF DOCUMENT AUTHENTICATION	78
3.3.7 VERIFICATION OF DOCUMENT AUTHENTICATION	85
3.3.7.1 USE CASE DIAGRAM OF VERIFICATION OF DOCUMENT AUTHENTICATION.....	85
3.3.7.2 CLASS DIAGRAMS OF VERIFICATION OF DOCUMENT AUTHENTICATION.....	86
3.3.7.3 SEQUENCE DIAGRAM OF VERIFICATION OF DOCUMENT AUTHENTICATION.....	89
3.4 DEVELOPMENT OF APPLICATION PROTOTYPE.....	95
3.4.1 IMPORT DIGITAL CERTIFICATE CODIFICATION	96
3.4.2 DOCUMENT SIGNING CODIFICATION	98
3.4.3 DOCUMENT AUTHENTICATION CODIFICATION.....	99
3.4.4 VERIFICATION OF DOCUMENT AUTHENTICATION CODIFICATION.....	102
4 RESULTS AND DISCUSSIONS	104
4.1 IDENTITY ASSOCIATION.....	104
4.1.1 ADVANTAGE OF DIGITAL CERTIFICATES.....	104
4.1.2 DISADVANTAGE OF DIGITAL CERTIFICATES.....	104
4.2 REVOCATION LIST (RL)	105
4.2.1 ISSUE WITH DIGITAL IDENTITIES	106
4.3 BLOCKCHAIN TIMESTAMP PERFORMANCE	106
4.3.1 AVERAGE HASHRATE PER DAY	107
4.3.2 AVERAGE BLOCK TIMESTAMP INTERVALS	108
4.3.3 AVERAGE TRANSACTION FEE	108
5 CONCLUSIONS.....	111

5.1 REGARDING RESEARCH OBJECTIVES.....	111
5.2 FINAL CONSIDERATIONS	113
5.3 RECOMMENDATIONS FOR FUTURE WORK	114
REFERENCES.....	116
APPENDIX 1 – MAINACTIVITY OF ANDROID APP.....	124
APPENDIX 2 – SIGNERASYNCCTASK OF ANDROID APP	136
APPENDIX 3 – OPRETURNMAIN OF JAVA WEB APPLICATION SERVER	139
APPENDIX 4 – VERIFYNOTARIZATIONASYNCCTASK OF ANDROID APP	145

1 INTRODUCTION

The emergence of the bitcoin system, result of the convergence of twenty years of research in distributed systems and currencies, has brought a revolutionary technology (ANTONOPOULOS, 2014, p. 219): “A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution” (NAKAMOTO, 2008, p. 1).

The bitcoin system brought a concept called decentralized trust, which differs from the traditional banking and payment system (ANTONOPOULOS, 2014, p. 15). In Brazil, for example, trust is centralized on the authority of the Brazilian Central Bank. In bitcoin, there is no central authority or point of control (ANTONOPOULOS, 2014, p. 3). The “central authority is replaced by consensus algorithms established among users participating in a network” (DINIZ, 2018, p. 51), so trust is achieved through the peer-to-peer network of the decentralized bitcoin system.

Sharing of computer hardware resources such as computational processing, hard disk storage and internet connection constitutes the architecture of peer-to-peer network, where each computer simultaneously provides and consumes resource-sharing services without the existence of central entities. Currently, the peer-to-peer network is mostly used to provide file and music file sharing services (SCHOLLMEIER, 2001, p. 101).

In the peer-to-peer network of the bitcoin system, each participating computer in the network shares a copy of a ledger database called blockchain. The blockchain concept emerged from a timestamp data structure that evidences the occurrence of payment transactions at a date and time in the past in the bitcoin system.

Although blockchain was primarily intended to record bitcoin transactions, public and private institutions are using it as a technology to provide services in sectors, such as health, real estate registration and government entrepreneurship.

In healthcare, Factom startup is using the blockchain technology to provide a cataloging system for medical records where personal health information is encrypted and then registered in the blockchain, protecting the confidentiality of patients (STRAY, 2019).

In real estate registration, a bitcoin company called BitFury in partnership with the government of the Republic of Georgia developed a real estate registration

system using blockchain to increase property ownership transparency and reduce the prevalence of fraud (HIGGINS, 2018). By April 2016, more than 100,000 properties in the Republic of Georgia had been registered in the blockchain, making it impossible for property records to be manipulated (SMERKIS, 2019).

In government entrepreneurship, World Economic Forum's Center for the Fourth Industrial Revolution, which is an international non-profit public-private hub for global impact, is developing projects to verify the potential of blockchain technology in "Interoperability, integrity, and inclusion: Blockchain for supply chains; Central banks in the age of blockchain; Unlocking transparency; Re-imagining data ownership and economic models in the token economy" (WORLD ECONOMIC FORUM, 2017, p. 13).

1.1 CONTEXT

There is a lot of interest in using blockchain technology to provide services in different sectors. In Brazil, blockchain can provide a document timestamping service for documents signed with Brazilian-Public Key Infrastructure (ICP-Brasil) certificates.

Currently, there are two notarial systems in Brazil. The first notarial system is the traditional model which is exercised by notary officers, who enter the notarial activity through public tender issued by a government procuring authority (BRASIL, 1988).

The second notarial system is the digital certification model implemented by ICP-Brasil Standards, which is exercised by Certification Authorities (CA), Registration Authorities (RA) and Timestamp Authorities (TA) (ITI. Entes da ICP-Brasil, 2019).

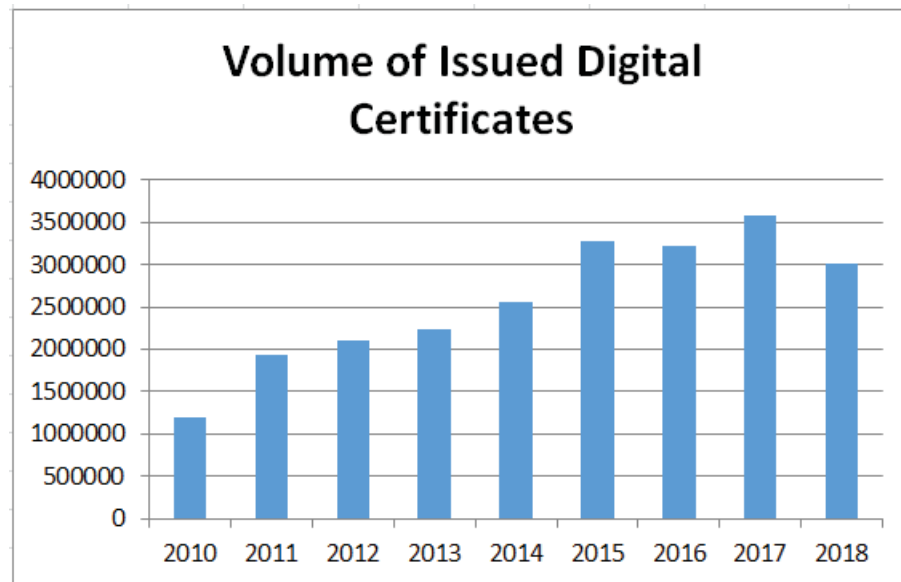
The ICP-Brasil was initiated in 2001 through Provisional Measure 2.200-2/2001, and is subordinated to the Instituto Nacional de Tecnologia da Informação (ITI), which is a federal autarchy, associated with the Civil House of the Presidency of the Republic that maintains and implements the ICP-Brasil policies (ITI, 2020g).

1.1.1 PESTLE

A PESTLE (Political, Economic, Social, Technological, Legal, and Environmental) analysis on ICP-Brasil certification points out the growing volume of

issued digital certificates (CHART 1) in Brazil is due to federal regulations that require the use of digital certificates in various public services.

CHART 1 – DIGITAL CERTIFICATES ISSUED YEAR BY YEAR



SOURCE: ITI (2020h).

There are three examples of mandatory provisional measures that have increased the use of digital certificates:

1. the provisional measure that came into force on 7/1/2018 requires “companies with employees and annual turnover below R\$ 78 million, including Micro Empreendedores Individuais (MEIs - individual microentrepreneurs)” to “declare their payroll information via e-Social¹, using the digital certificate ... It is estimated that around 20 million companies will be subject to this obligation” (ITI, 2020c);
2. the provisional measure that came into force on 10/01/2018 requires companies with annual turnover above R\$ 78 million to provide information to the government about their employees through the

¹ eSocial is an online system that “employers communicate to the Government information related to workers, such as bonds, social security contributions, payroll, work accident communications ...” (eSocial. Conheça o eSocial, 2019).

eSocial system, which requires the use of a digital certificate to access the system (ID SEGURO, 2019);

3. the provisional measure that came into force on 01/04/2015 requires all employers to use a digital certificate to access the Empregador (Employer) Web system, in order to inform "the Ministry of Labor and Employment (Ministério do Trabalho e Emprego) of employee dismissal for the purpose of receiving Unemployment Insurance" (ITI, Pedido de Seguro-Desemprego, 2019).

In August 2018, there was a historical record of 504,000 issued digital certificates. This record was driven mainly by the provisional measure that came into force on 7/1/2018, requiring "companies with employees and annual turnover below R\$ 78 million to declare their payroll information via eSocial, with use of the digital certificate" to access the system (ITI, 2020c).

In January 2018, a provisional measure came into force that "requires companies with annual turnover above R\$ 78 million to provide information to the government about their employees through the eSocial system" (ID SEGURO, 2019). In that month there were 331,000 issued digital certificates. And in the months that followed up, there were continued increases in the volume of issued digital certificates that culminated in the historical record of 504,000 issued digital certificates in August 2018.

This month-by-month sequence in the volume of issued digital certificates in 2018 was the result of a combination of the two provisional measures mentioned above that required companies to use a digital certificate to access the eSocial system in order to provide employee information.

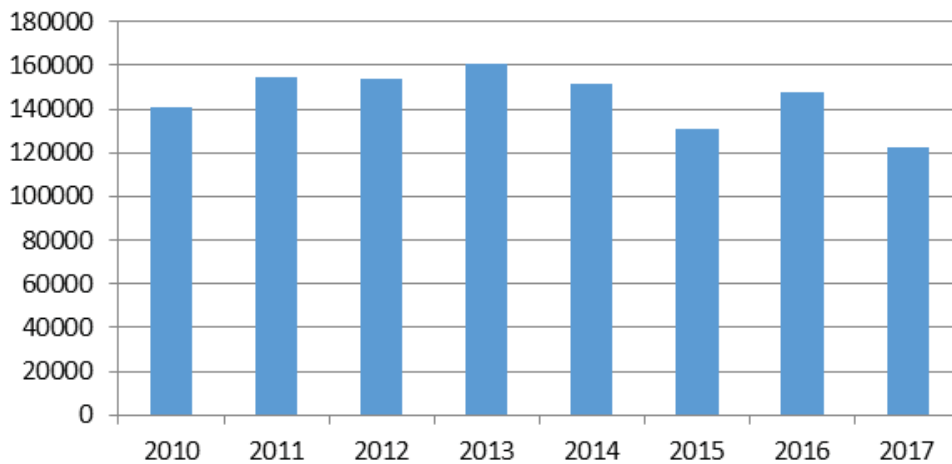
In June 2015, there were 450,000 issued digital certificates. This volume only lags behind to the historical record of 504,000 issued digital certificates in August 2018. This high volume of issued digital certificates in June 2015 was influenced by the provisional measure that came into force in April 2015, requiring all employers to use a digital certificate to access the Empregador Web system to inform "the Ministry of Labor and Employment of employee dismissal for the purpose of receiving Unemployment Insurance" (ITI, Pedido de Seguro-Desemprego, 2019).

By analyzing CHART 1, it is clear there was a year-by-year increase in the volume of issued ICP-Brasil digital certificates. Nevertheless, this year-by-year

increase has occurred concurrently with continued years of economic instability. Between 2010 and 2017, the lowest level of income tax revenues of legal entities occurred in 2017 as indicated by the Consumer Price Index (IPCA), with R\$ 122 billion in income tax revenues. However, in 2017, there was a historical record of 3.5 million issued ICP-Brasil digital certificates up until 2017 (CHART 2).

CHART 2 – TOTAL LEGAL ENTITY INCOME TAX COLLECTION – IPCA INDEX

TOTAL LEGAL ENTITY INCOME TAX COLLECTION - IPCA (IN MILLIONS)



SOURCE: RECEITA FEDERAL (2018).

It is possible to notice that there is a contrast between the increase in the volume of issued ICP-Brasil digital certificates and the decrease in income tax revenues of legal entities (TABLE 1). A simple linear regression analysis was performed in order to verify if there was any impact on the use of ICP-Brasil digital certificates in order to increase income tax revenues of legal entities between 2010 and 2017.

TABLE 1 – COMPARISON BETWEEN TOTAL ISSUED DIGITAL CERTIFICATES AND TOTAL INCOME TAX COLLECTION

Year	Total Issued Digital Certificates	Total Legal Entity Income Tax Collection - IPCA (in millions)
2010	1,204,095	141,163
2011	1,937,198	154,892
2012	2,107,225	153,712
2013	2,229,440	161,048
2014	2,553,708	151,712
2015	3,280,537	130,699
2016	3,225,560	147,943
2017	3,587,709	122,434


















Source: ITI (2020h); RECEITA FEDERAL (2018).

At 5% significance level, it was verified that the use of ICP-Brasil digital certificates did not impact on the increase of income tax revenues of legal entities between 2010 and 2017 since the analysis indicated an impact of only 20% on the use of ICP-Brasil digital certificates on the increase of income tax revenues of legal entities.

The increase in the use of ICP-Brasil digital certificates is due to federal government provisional measures that require companies to use ICP-Brasil digital certificates to provide information about their employees through systems such as eSocial. This year-on-year increase in emissions of ICP-Brasil digital certificates occurs in a scenario of economic instability with the low levels of income tax collections in recent years.

In regards to cryptocurrency adoption in Brazil, a “Report on International Bitcoin Flows 2013 – 2019” (CRYSTAL, 2019) compiled by Crystal, a blockchain analytics platform, Brazil is ranked 9th among countries with the most cryptocurrency exchanges, currently having 7 active exchanges: MercadoBitcoin, FoxBit, BitcoinToYou, NegocieCoins.com.br, CoinBene, Braziliex, FlowBTC. The country with the most exchanges is the European Union with 49 cryptocurrency exchanges, followed by the United Kingdom (43 exchanges) (FIGURE 1).

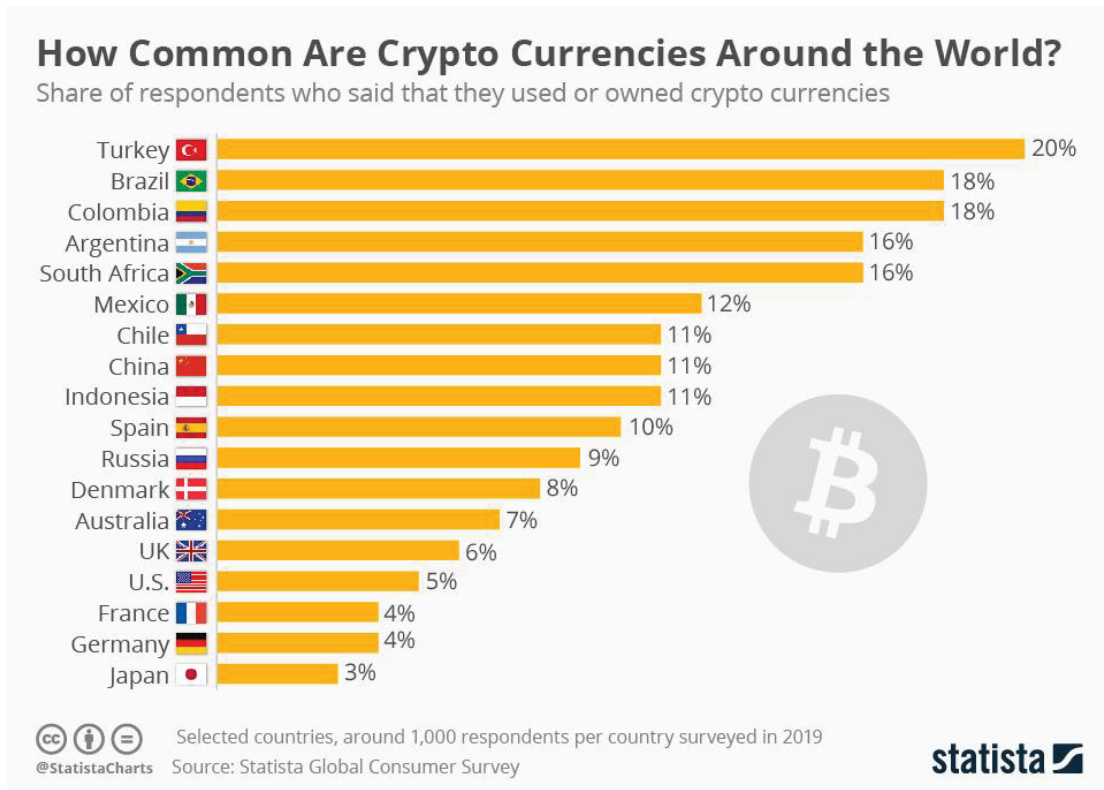
FIGURE 1 – CRYPTOCURRENCY EXCHANGES BY COUNTRIES

Country of Registration	Number of Exchanges
 European Union	49
 UK	43
Unknown	33
 USA	27
 Hong Kong	22
 Singapore	19
 Australia	9
 China	9
 Brazil	7
 South Korea	7
 Japan	6
 Canada	6
 Turkey	4
 Argentina	3
 India	3
 Mexico	3
 Russia	3
 Indonesia	1

SOURCE: CRYSTAL (2019).

According to Statista (2019), a market and consumer data provider, Latin America is the world region with the most cryptocurrency users, with Brazil and Colombia leading the region. Worldwide, Brazil is the second country with most cryptocurrency users, behind Turkey (FIGURE 2).

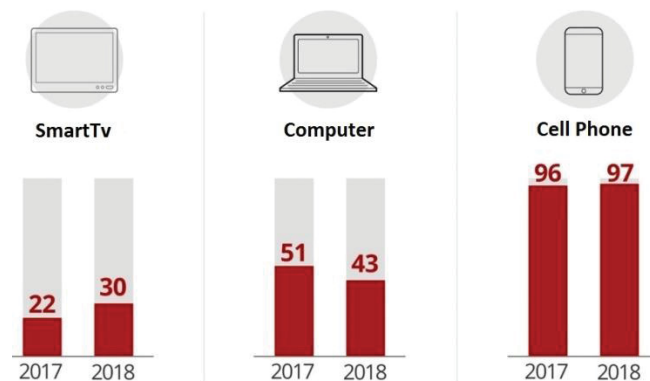
FIGURE 2 – COUNTRIES WITH MOST CRYPTOCURRENCY USERS



SOURCE: STATISTA (2019).

The increase in the use of ICP-Brasil certificates and cryptocurrencies in Brazil is also coupled with an increase in internet access among the Brazilian population. According to G1 (2019), 70% of the Brazilian population (126.9 million people) used the internet regularly in 2018. 97% of the 126.9 million access the internet through cell phones (FIGURE 3)

FIGURE 3 – DATA OF INTERNET USE IN BRAZIL



SOURCE: G1 (2019).

1.2 JUSTIFICATION

This research project aligns with the Information Management Postgraduate Programme of the Universidade Federal do Paraná in regards to its Information and Technology line of research that studies “techniques and tools with a view to transforming data and information as an input for the development and improvement of technological processes and products” (UNIVERSIDADE FEDERAL DO PARANÁ, 2020). This project proposes the use of blockchain timestamping and digital signatures based on ICP-Brasil digital certificates to provide a ubiquitous digital document authentication service through an application prototype for mobile Android devices.

It is intended to solve the problem of increased demand to digitalize the notarial system in Brazil since mobile cellular devices introduced digital ubiquity in Brazil, especially now that 97% of Brazilian internet users access the internet through cell phones (G1, 2019). A recent example of this increased demand is due to the COVID-19 pandemic that has spread in Brazil, which has prompted the Health Ministry through Government Ordinance 467/20 (MINISTÉRIO DA SAÚDE, 2020) to allow doctors to make medical appointments via online videoconferencing and issue digital medical prescriptions signed with ICP-Brasil digital certificates for “special control drugs and prescriptions for antimicrobials” (ANVISA, 2020).

Documents signed with ICP-Brasil digital certificates have legal validity equivalent to a document signed on paper (ITI, 2020b). Additionally, any form of electronic document admitted by the parties also has legal validity equivalent to a document signed on paper according to paragraph 2 of Provisional Measure 2.200-2/2001:

Paragraph 2. The provisions of this Provisional Measure do not preclude the use of any other means of proving the authorship and integrity of documents in electronic form, including those that use certificates not issued by ICP-Brasil, as long as admitted by the parties as valid or accepted (BRASIL, 2001).

Additionally, blockchain can timestamp documents signed with ICP-Brasil digital certificates in order to add attributes of date and time to a signed document (ITI, 2019) and provide data integrity “due the existence of a long chain of blocks [that] makes blockchain’s deep history immutable, [which is] a key feature of bitcoin’s

security” (ANTONOPOULOS, 2014, p. 164) and its decentralized peer-to-peer network distributed worldwide.

1.3 OBJECTIVES

The objectives of this research project are divided in general and specific objectives.

1.3.1 GENERAL OBJECTIVE

This research aims to integrate blockchain timestamping and digital signatures based on ICP-Brasil digital certificates to provide a digital document authentication service.

1.3.2 SPECIFIC OBJECTIVES

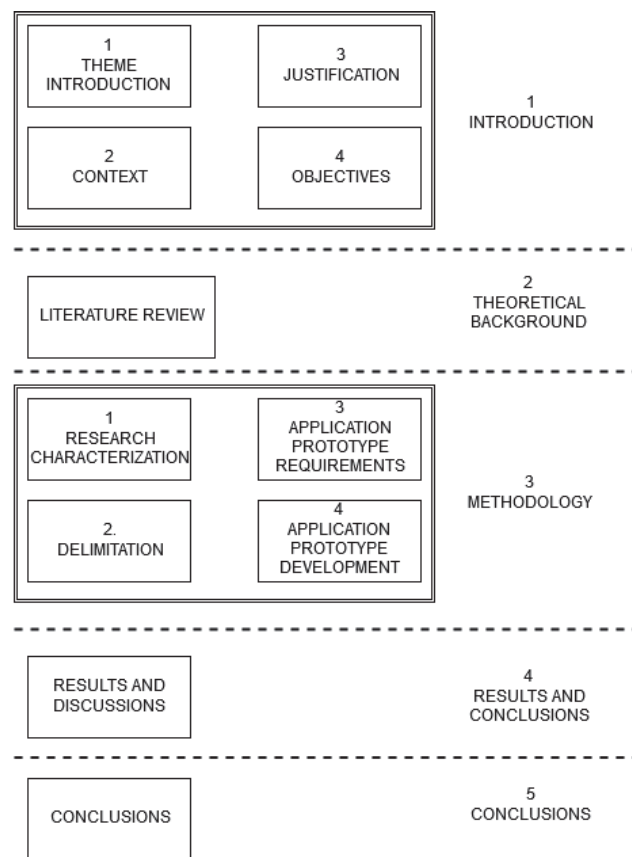
- a) Identify how integration of blockchain timestamping and digital signatures based on ICP-Brasil digital certificates assures information security considering concepts of authenticity, availability, confidentiality, identity, immutability, integrity, legality and non-repudiation.
- b) Identify system requirements to develop an application prototype for mobile Android devices that blockchain timestamping and digital signatures based on ICP-Brasil digital certificates to provide a digital document authentication service.
- c) Develop an application prototype for mobile Android devices that integrates blockchain timestamping and digital signatures based on ICP-Brasil standards.

1.4 THESIS STRUCTURE

This thesis is structured in five chapters (FIGURE 4). Chapter 1 presents the context, justification, and objectives (general and specific). Chapter 2 begins with the introduction of the bitcoin cryptocurrency, and its operation as “an electronic payment system based on cryptographic proof” (NAKAMOTO, 2008, p. 1). It is followed by a

conceptualization of blockchain's data structure, which includes the application of the SHA 256 algorithm and the use of Merkle Tree. Chapter 3 presents the research characterization, research geographical delimitation, application prototype requirements, and application prototype development. Chapter 4 presents results and discussions regarding achieving the general and specific objectives proposed in this research project. Chapter 5 presents conclusions regarding this research project's application prototype.

FIGURE 4 – FLOWCHART OF THESIS STAGES



SOURCE: AUTHOR (2019).

2 LITERATURE REVIEW

This literature review aims to present the topics that form the basis of the research.

2.1 BITCOIN CRYPTOCURRENCY

“With the publication of the Bitcoin white paper in 2008, and the subsequent delivery of a first prototype implementation of Bitcoin two months later, the individual or group behind the alias “Satoshi Nakamoto” was able to forge a new class of decentralized currency [...] relying on basic cryptographic constructs, such as hash functions and digital signatures” (KARAME; ANDROULAKI, 2016, p. 1).

In the book titled “*Bitcoin and Blockchain Security*”, Karame *et al.* present the concept of the bitcoin cryptocurrency system:

The design of Bitcoin offered the world a promise for a low-cost decentralized and anonymous currency. The core idea of Bitcoin is simple. The system allows two or more parties to exchange financial transactions without passing through intermediaries (such as banks or payment processors). These transactions are validated collectively in a peer-to-peer network by all users. This not only eliminates the need for centralized control (e.g., by banks), but also reduces the cost of making transactions (at the national and international levels). [...] Bitcoin does not require users to register their identity/credentials nor does it require them to fill out endless forms in order to set up an account (KARAME; ANDROULAKI, 2016, p. 1).

2.1.1 BLOCKCHAIN

A blockchain or timestamp server evidences that a transaction occurred in a particular date and time in the past (ITI, 2020d). In the blockchain data structure, each block stores accounting information for a certain amount of bitcoin transactions. A chain of blocks is formed by identifying each block in the blockchain by an identifier (ANTONOPOULOS, 2014, p.163).

In the bitcoin cryptocurrency system, a node is a computer participating in the peer-to-peer network. Each node shares a copy of a timestamp server, known as blockchain. Thus, simultaneously each participating computer node tracks a single bitcoin transaction, and watches that transaction become trusted and accepted by the bitcoin distributed consensus mechanism until it is registered in the bitcoin blockchain (ANTONOPOULOS, 2014, p. 15). Nodes are “aware of all transactions

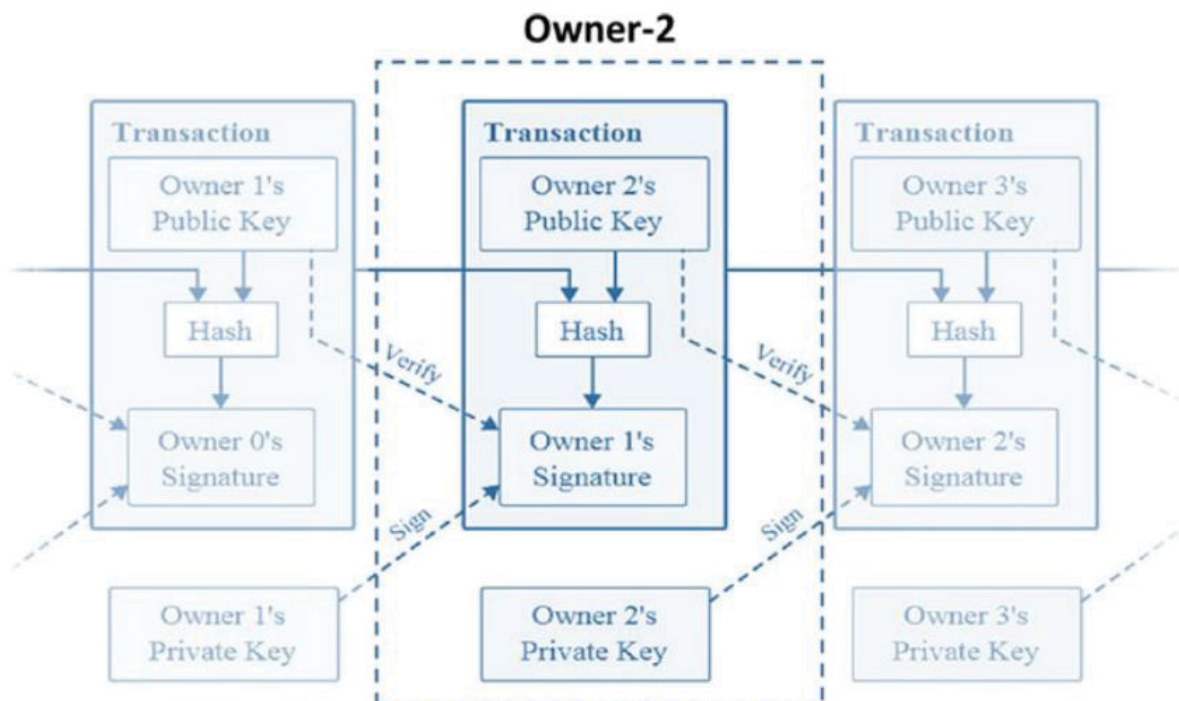
that happened in the past. [...] So, the nodes that would validate the transactions should definitely be accessible to the whole blockchain data since the genesis block” (SINGHAL; DHAMEJA; PANDA, 2018, p. 126).

2.1.2 TRANSACTIONS

Bitcoin transactions are “a chain of digital signatures” (NAKAMOTO, 2008, p. 2) (FIGURE 5). According to Karame *et al.* a single bitcoin transaction is:

formed by digitally signing a hash of the previous transaction where this coin was last spent along with the public key of the future owner and incorporating this signature in the coin. Transactions take as input the reference to an output of another transaction that spends the same coins and output the list of addresses that can collect the transferred coins. A transaction output can only be redeemed once, after which the output is no longer available to other transactions. Once ready, the transaction is signed by the user and broadcast in the P2P network. Any peer can verify the authenticity of a BTC by checking the chain of signatures (KARAME; ANDROULAKI, 2016, p. 34).

FIGURE 5 – CHAIN OF DIGITAL SIGNATURES



SOURCE: SINGHAL *et al.* (2018, p. 180).

Authors of book “*Blockchain: A Beginner’s Guide to Building Blockchain Solutions*”, Singhal *et al.* (2018) detail how a bitcoin transaction works:

Notice only the highlighted Owner-2 section in diagram (FIGURE 5). Since Owner-1 is initiating this transaction, he is using his private key for signing the hash of two items: one is the previous transaction where he himself received the amount and the second is Owner-2's public key. This signature can be easily verified using the public key of Owner-1 to ensure that it is a legitimate transaction. Similarly, when Owner-2 will initiate a transfer to Owner-3, he will use his private key to sign the hash of the previous transaction (the one he received from Owner-1) along with the public key of Owner-3. Such a transaction can be, and will be, verified by anyone who is a part of the network. Obviously because every transaction is broadcast, most of the nodes will have the entire history of transactions to be able to prevent double-spend attempts. (SINGHAL; DHAMEJA; PANDA, 2018, p. 180-181).

2.1.3 DOUBLE-SPENDING

To exemplify a double-spending attack in the blockchain, Tschorsch *et al.* (2016) describe the following scenario:

(i) broadcast a regular transaction (e. g., paying for a product), (ii) secretly mine on a fork which builds on the last block and includes a conflicting transaction which uses the same outputs as in step (1), but pays the attacker instead of the seller, (iii) wait until the seller is confident (i.e., receives enough confirmations) and hands the product over, (iv) as soon as the secret fork is longer than the public chain, broadcast the respective blocks. Because the secret branch is longer, the network will consider it as the valid main block chain. The regular transaction becomes invalid and cannot (even when broadcast by the seller) be included in a block anymore (TSCHORSCH; SCHEUERMANN, 2016, p. 9).

Double-spending prevention is possible through the creation of a blockchain along with a system of proof-of-work.

2.1.4 PROOF-OF-WORK

Through a system of proof-of-work, it is possible to “achieve consensus without a central trusted authority” (ANTONOPOULOS, 2014, p. 4).

In a decentralized network, someone has to be selected to record the transactions. The easiest way is random selection. However, random selection is vulnerable to attacks. So if a node wants to publish a block of transactions, a lot of work has to be done to prove that the node is not likely to attack the network. Generally the work means computer calculations (ZHENG, Z. *et al.*, 2017, p. 559).

“Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it” (NAKAMOTO, 2008, p. 2). Growth of the blockchain is based on majority of CPU processing power, which “will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes” (NAKAMOTO, 2008, p. 2).

The theory behind proof-of-work is that representation by majority of CPU processing power is more secure than representation by “one-IP-address-one-vote” since anyone could allocate IP addresses to control the blockchain, corrupting it with double-spending (NAKAMOTO, 2008, p. 2) (TSCHORSCH; SCHEUERMANN, 2016, p. 3).

To increase the growth of the blockchain participating nodes having the highest CPU processing power compete to timestamp blocks to be added to the blockchain by solving a cryptographic puzzle that consists of repeatedly calculating by trial and error a block header’s SHA 256 Hash.

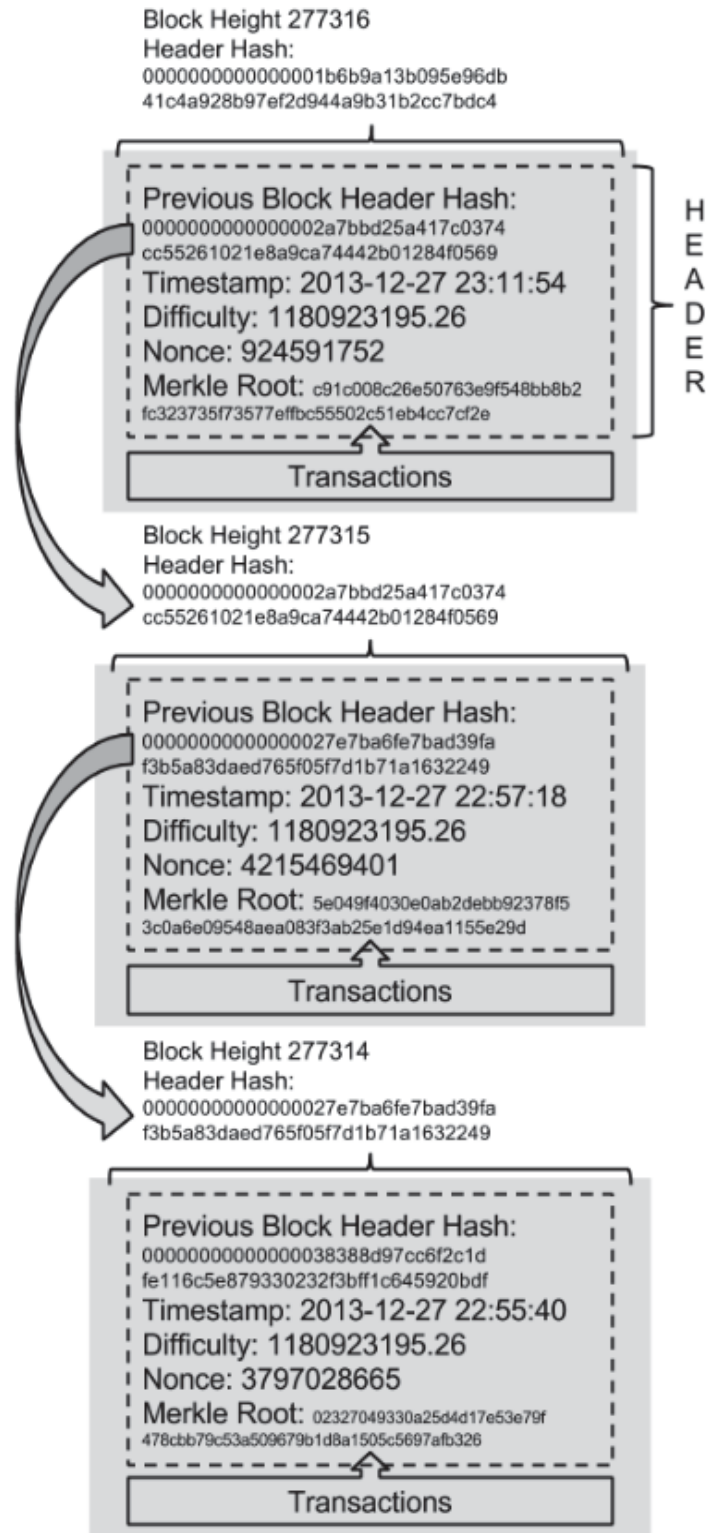
For every calculation attempt, a 4-byte metadata called *Nonce* located in the block header is randomly modified by trial and error until the resulting calculation of the block header is lower or equal a pre-determined target value (TSCHORSCH; SCHEUERMANN, 2016, p. 3) (ANTONOPOULOS, 2014, p. 27).

The process of competing to timestamp blocks to be added to the blockchain is called mining. Nodes competing to mine blocks receive new minted coins as reward for securing the blockchain (TSCHORSCH; SCHEUERMANN, 2016, p. 3).

In the bitcoin blockchain data structure (FIGURE 6):

each block within the blockchain is identified by a hash, generated using the SHA256 cryptographic hash algorithm on the header of the block. Each block also references a previous block, known as the parent block, through the “previous block hash” field in the block header. In other words, each block contains the hash of its parent inside its own header. The sequence of hashes linking each block to its parent creates a chain going back all the way to the first block ever created, known as the genesis block (ANTONOPOULOS, 2014, p. 163).

FIGURE 6 – BLOCKS LINKED IN A CHAIN TO THE PREVIOUS BLOCK HEADER HASH



SOURCE: ANTONOPOULOS (2010, p. 169).

2.1.5 BLOCK HEADER

The block header is composed of three sets of metadata (FIGURE 6). The first set is the *Previous Block Hash*, which connects a block to a previously created block in the blockchain. The second set refers to a block's *Timestamp*, *Difficulty* and *Nonce*. The third is the *Merkle Tree Root*, which is a 32-byte hash obtained from a double calculation of the Merkle Tree using the SHA 256 algorithm (ANTONOPOULOS, 2014, p. 165).

There are two ways to identify a block, the primary identifier is obtained by calculating a "block header twice through the SHA 256 algorithm" (ANTONOPOULOS, 2014, p. 165). For example:

$$\text{BlockHash} = \text{SHA256}(\text{SHA256}(\text{Header}))$$

which results in a 32-byte hash, called a block header hash. As an example, the "first block ever created" or genesis block is 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f (ANTONOPOULOS, 2014, p. 165).

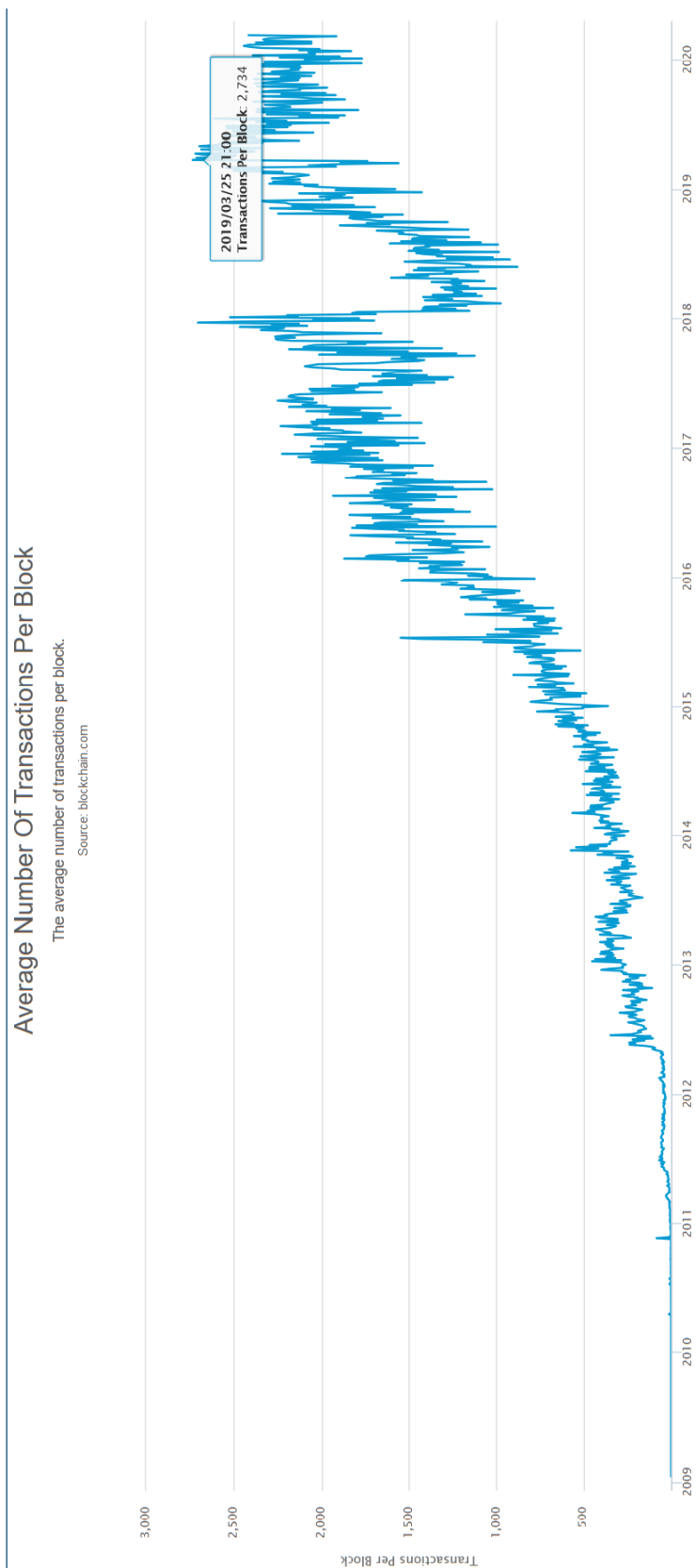
The second way to identify a block is through its height. "The first block ever created is at block height 0 (zero)" (ANTONOPOULOS, 2014, p. 165).

2.1.6 TIMESTAMPING IN THE BLOCKCHAIN

"Bitcoin's blocks [that] are generated every 10 minutes, on average" (ANTONOPOULOS, 2014, p. 199). Each newly generated block registers a timestamp in its the block header (SECTION 2.1.5; FIGURE 6) in Unix time format (SINGHAL; DHAMEJA; PANDA, 2018, p.126), which tracks time in number of seconds between a particular date and the Unix Epoch that started counting on January 1st, 1970 at Coordinated Universal Time – UTC (UNIX TIMESTAMP, 2020).

Bitcoin blocks size are of 1 MB (SINGHAL; DHAMEJA; PANDA, 2018, p.161). The highest average number of transactions per block ever recorded occurred on March 25, 2019 with a total of 2,671 transactions per block (FIGURE 7).

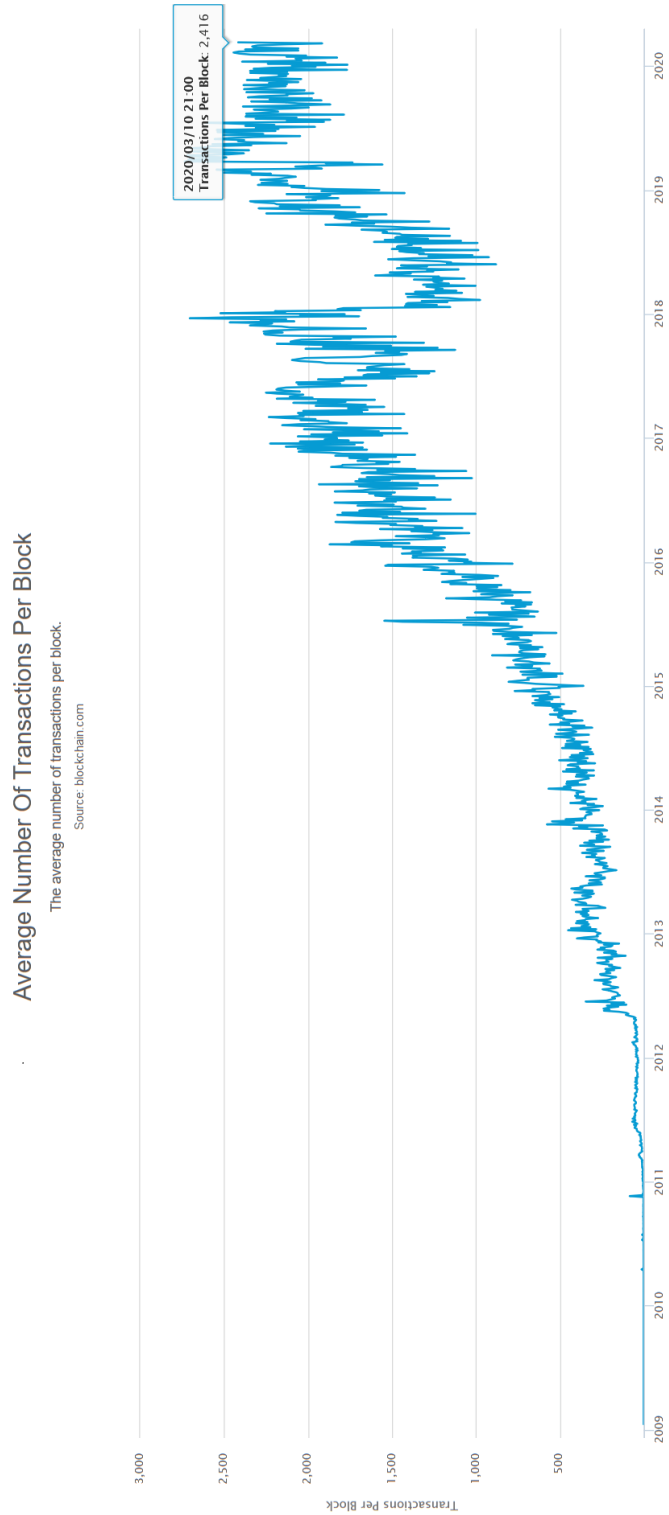
FIGURE 7 – BITCOIN HIGHEST AVERAGE NUMBER OF TRANSACTIONS PER BLOCK



SOURCE: BLOCKCHAIN. Average Number of Transactions per Block (2020).

On March 11, 2020, the average of number of transactions per block was 2,318 (FIGURE 8).

FIGURE 8 – BITCOIN AVERAGE NUMBER OF TRANSACTIONS PER BLOCK ON MARCH 11, 2020



SOURCE: BLOCKCHAIN. Average Number of Transactions per Block (2020).

2.2 SHA 256 ALGORITHM

The SHA 256 algorithm has important application in the bitcoin system's operations. Therefore, it is presented specifications that define the standard of a secure SHA 256 algorithm.

How do you prove that a document was not modified? By calculating the SHA 256 algorithm from a file, it results in a type of a "DNA" of the file (ORIGINALMY, 2020), called message digest, which is a condensed representation of electronic data (FIPS PUB 180-4, 2018, p. 3) of "fixed-size result" (FERGUSON; SCHNEIER, 2003, p. 83). To prove it was not modified, a message digest is recalculated from the file. Then, the message digests are compared to detect whether the file was changed (FIPS PUB 180-4, 2018, p. iii).

This action of generating a message digest from a file is analogous to an authentication of a document, which in the Brazilian notarial system "consists of the notary's recognition of a document as being authentic or true" (DICIO, 2018).

The document "FIPS PUB 180-4" was written by the US National Institute of Standards and Technology (NIST), who publishes standardization documents, called Federal Information Processing Standards (FIPS). FIPS describes "document processing, encryption algorithms and other information technology standards for use within non-military government agencies and by government contractors and vendors who work with the agencies" (FIPS. What is FIPS, 2019).

Additionally, the "FIPS PUB 180-4" specifies several functions of secure hash algorithms, such as: "SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256" (FIPS PUB 180-4, 2018, p. 3). "The message digests range in length from 160 to 512 bits, depending on the algorithm" (FIPS PUB 180-4, 2018, p. iv). In SHA 256 algorithm, the length of message digests is 256 bits.

The several SHA algorithm functions differ mostly in security strengths (FIPS PUB 180-4, 2018, p. 3). Larger SHA algorithms result in larger block sizes and larger size of message digests (FIPS PUB 180-4, 2018, p. 3).

For each SHA algorithm there are two stages: "preprocessing and hash computation. Preprocessing involves padding a message, parsing the padded message into m-bit blocks, and setting initialization values to be used in the hash computation. [...] The final hash value generated by the hash computation is used to determine the message digest" (FIPS PUB 180-4, 2018, p. 3).

As mentioned earlier, the message digest length in each m -bit block in the SHA 256 algorithm is 512 bits, which is represented by a sequence of sixteen 32-bit words.

Any SHA algorithm listed in TABLE 2 is considered safe as it is computationally unfeasible:

to find a message that corresponds to a given message digest, or 2) to find two different messages that produce the same message digest. Any change to a message will, with a very high probability, result in a different message digest (FIPS, 2018, p. iv).

TABLE 2 – PROPERTIES OF SHA HASH ALGORITHMS

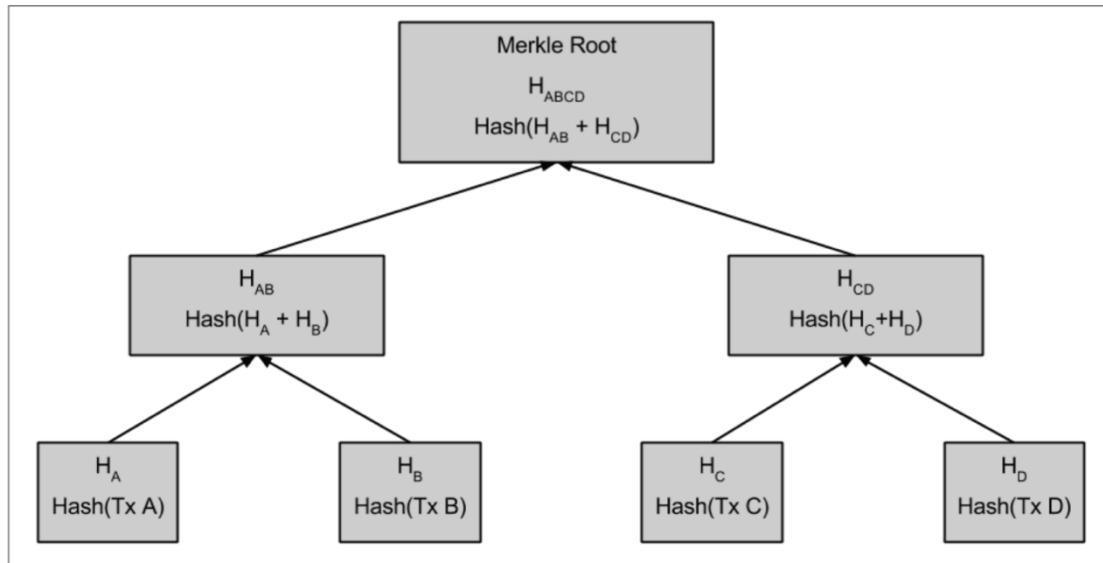
Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

SOURCE: FIPS (2018, p.3).

In the bitcoin system, the calculation of the SHA 256 algorithm, besides being applied to mining for proof-of-work, is applied to the creation of *Merkle Trees*, which is a binary tree that contains cryptographic calculations of the SHA 256 algorithm, and is also known as Binary Hash Tree.

A Merkle Tree is a structure used to summarize and verify the integrity of large amounts of data (FIGURE 9). In the blockchain, the Merkle Tree is used to represent a digital fingerprint of all transactions in each block (ANTONOPOULOS, 2014, p. 170). It is constructed by recursively calculating the pairs of nodes in the merkle tree through the SHA 256 algorithm until there is only one hash, called root, or merkle root (ANTONOPOULOS, 2014, p. 170).

FIGURE 9 – CALCULATING NODES IN A MERKLE TREE



SOURCE: ANTONOPOULOS (2010, p. 171).

2.3 PRIVATE-KEY OR SYMMETRIC-KEY ENCRYPTION

In private-key or symmetric-key encryption, also known as classical cryptography, according to Jonathan *et al.* (2015, p. 4), the objective is to enable two parties to communicate secretly in the presence of an eavesdropper. A message sender encrypts plaintext using a secret key and sends the resulting ciphertext to a message receiver, who also knows the secret key and can decrypt the ciphertext back into plaintext. Decryption is identical to encryption in private-key cryptography since both message sender and message receiver share a common secret key. The only difference is that the key schedule is reversed (STINSON, 1995, p. 114).

According to Jonathan *et al.* (2015, p. 7), the cryptographic method is not required to be secret, but security should only rely on the confidentiality of the secret key. The drawback of a private-key cryptosystem is that it renders the system insecure since it requires prior communication of a secret key between message sender and message receiver (STINSON, 1995, p. 114). To circumvent this security issue, communication of a secret key can be performed through confidentiality and authentication schemes implementations of public-key or asymmetric-key encryption (STALLINGS, 2014, p. 430), which are described in section 2.4.

2.4 PUBLIC-KEY OR ASYMMETRIC-KEY ENCRYPTION

In public-key or asymmetric-key encryption, each user in the system generates locally on a computer a pair of related keys – a public-key and a private-key (secret-key). Only the public-key is published to all participants in the system. And the private-key (secret-key) is kept private, never distributed and remains protected (STALLINGS, 2014, p. 258).

“The idea behind a public-key system is that it might be possible to find a cryptosystem where it is computationally infeasible to determine” a private-key (secret-key) given a public-key (STINSON, 1995, p. 114). Or, in other words, “it should not be possible to compute the secret key from the corresponding public key” (FERGUSON; SCHNEIER, 2003, p. 27).

According to Stallings (2014, p. 256), public-key encryption was created to solve two problems: private-key (secret-key) distribution and digital signatures. Additionally, the use of public-key and private-key (secret-key) has important consequences in the areas of confidentiality and authentication (STALLINGS, 2014, p. 254).

The pair of related keys is used “to perform complementary operations, such as encryption and decryption or signature generation and signature verification” (STALLINGS, 2014, p. 255). A private-key (secret-key) can encrypt a message and a public-key is used to decrypt a message previously encrypted by the private-key (secret-key); equally, a public-key can encrypt a message and a private-key (secret-key) is used to decrypt a message previously encrypted by the public-key (DAVIES, 2011, p. 157).

It doesn't matter which one of the pair of related keys (public-key or private-key) is used to encrypt a message, i.e., if a message is encrypted with A-key, the message should be decrypted with related B-key, and vice-versa (DAVIES, 2011, p. 157). But encrypting a message with either public-key or private-key (secret-key) enables different cryptographic scheme implementations (STALLINGS, 2014, p. 260).

A cryptographic scheme implementation where A encrypts a message using a private-key (secret-key) and B decrypts the message with a related public-key serves as a digital signature and provides authentication since “only A could have prepared the message ... [and] it is impossible to alter the message without access

to A's private key, so the message is authenticated both in terms of source and in terms of data integrity" (STALLINGS, 2014, p. 260). The problem with this scheme is that it does not provide confidentiality because anyone with access to A's public-key can decrypt the message (STALLINGS, 2014, p. 261).

On the other hand, a cryptographic scheme implementation where A encrypts a message using a public-key and B decrypts the message with a related private-key (secret-key), this cryptographic scheme provides confidentiality because no other recipient can decrypt the message except B who is the only one who knows the private-key (secret-key) (STALLINGS, 2014, p. 258).

2.5 PUBLIC-KEY INFRASTRUCTURE

Even though "public-key cryptography makes the problem of distinguishing keys a lot simpler" (FERGUSON; SCHNEIER, 2003, p. 27) through the use of a pair of related keys – public-key and a private-key (secret-key), "the procedures involved are not simpler nor any more efficient than those required for [private-key or] symmetric encryption" (STALLINGS, 2014, p. 255), which renders the system insecure since it requires prior communication of a secret key between message sender and message receiver.

As a result, a key distribution protocol in the form of a public-key infrastructure (PKI) is needed to act as a central agent to allow users to "recognize which key belongs to whom" (FERGUSON; SCHNEIER, 2003, p. 315). A central agent prevents an impostor from impersonating another user since C can create a pair of related keys – a public-key and a private-key (secret-key) – and publish the public-key while impersonating B, so that A will think B's public-key belongs to B (FERGUSON; SCHNEIER, 2003, p. 29).

The central agent in a public-key infrastructure is called Certification Authority (CA). Through the public-key infrastructure, user A presents its own public-key to identify itself to the Certificate Authority, who, in turn, uses its private-key to sign a certificate that guarantees A's public-key belongs to A (FERGUSON; SCHNEIER, 2003, p. 29).

"In a PKI, each participant only has to have the CA certify his public key, and know the CA's public key so that he can verify the certificates of other participants" (FERGUSON; SCHNEIER, 2003, p. 29).

With the Certificate Authority's public-key, B verifies that the Certificate Authority has signed the public-key on A's certificate assuring that A's public-key belongs to A (FERGUSON; SCHNEIER, 2003, p. 316).

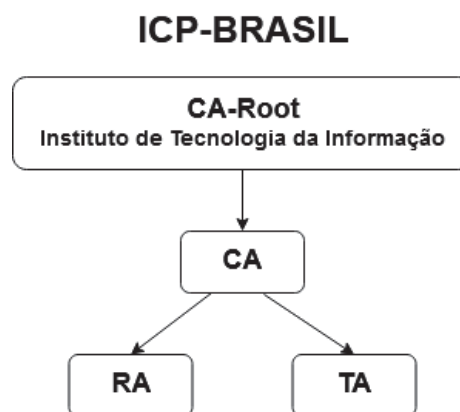
The problem in public-key infrastructure is that every user must be online in order to check for certificate revocation. A certificate revocation list (CRL) database operates to register a certificate that needs to be revoked when a user's private-key is compromised in case, for example, the user's computer gets hacked or stolen. Thus, besides verifying that a Certificate Authority has signed a certificate, a user must be online to also check the certificate revocation list database to verify if a certificate has been revoked (FERGUSON; SCHNEIER, 2003, p. 334).

2.6 BRAZILIAN PUBLIC KEY INFRASTRUCTURE (ICP-BRASIL)

There are two notarial systems in Brazil. The first system is the traditional model which is exercised by notary officers, who enter the notarial activity through public tender issued by a government procuring authority (BRASIL, 1988).

The second notarial system is the digital certification model implemented by ICP-Brasil Standards, which was initiated in 2001 through Provisional Measure 2.200-2/2001, and is subordinated to the Instituto Nacional de Tecnologia da Informação (ITI), which is a federal autarchy, associated with the Civil House of the Presidency of the Republic that maintains and implements the ICP-Brasil policies (ITI, 2020g).

FIGURE 10 – BRAZILIAN PUBLIC KEY INFRASTRUCTURE



SOURCE: AUTHOR (2019).

The ICP-Brasil Standards (FIGURE 10) is exercised by Certification Authorities (CA), Registration Authorities (RA) and Timestamp Authorities (TA). “ICP-Brasil’s Root Certification Authority (CA-Root) is the first authority in the certification chain” (ITI, 2020e). In the hierarchical chain, Certification Authorities (CA) are subordinated to the CA-Root. Under CAs are Registration Authorities (RA) and Timestamp Authorities (TA) (ITI, 2020e). Each entity in the hierarchical chain is described below as described in ITI’s official website:

Certification Authority (CA):

is an entity, public or private, [...] responsible for issuing, distributing, renewing, revoking and managing digital certificates. The Certification Authority (CA) is responsible for verifying that the certificate holder has the private key that corresponds to the public key that is part of the certificate. Creates and digitally signs a signer’s certificate, where the certificate issued by the CA represents an attestation of the holder's identity, which has a unique key pair (public/private) (ITI, 2020e). Examples of CAs: Serpro, Caixa Econômica Federal, Caixa Econômica Federal, Serasa Experian, Receita Federal do Brasil, Certisign.

Registration Authority (RA):

is responsible for the interface between the user and the Certification Authority. A Registration Authority (RA) is tied to a Certification Authority. Its purpose is to receive, validate, forward requests for issuance or revocation of digital certificates and identify, in person, claimant requests. It is the RA's responsibility to keep records of its operations. An RA may be physically located on a CA or it may be a remote registry entity (ITI, 2020e). Example of RA: Digital Sign RA.

Timestamp Authority (TA):

is an entity that users trust a Timestamp Authority (TA) to issue Timestamps. Timestamp Authorities are responsible for providing Timestamping services, which is a set of attributes provided by a TA that, when associated with a digital signature, proves its existence within a certain period of time. In practice, a document is created and its content is encrypted. Then it receives the attributes of year, month, day, hour, minute and second, attested in the form of a signature issued with a digital certificate thus serving to prove its authenticity. A TA attests not only the timestamp of a transaction, but also its content (ITI. Entes da ICP- Brasil, 2019). Example of TA: AC Caixa Timestamping.

2.7 INFORMATION SECURITY

There are three key principles of information security known as “confidentiality, integrity, and availability (CIA) triad” (ANDRESS, 2014, p. 5) that has been “the industry standard for computer security since the development of the mainframe” (WHITMAN; MATTORD, 2009, p. xviii). “But the C.I.A. triangle model is generally viewed as no longer adequate in addressing the constantly changing environment” (WHITMAN; MATTORD, 2009, p. 11). This security issue is caused by “increasing number of attacks — which are often able to bypass firewalls, intrusion prevention systems, and antivirus software — are aimed directly at web, mobile, and related applications” (BEAVER, 2016, p. 20). Therefore, other principles of information security such as authenticity, immutability, legality and non-repudiation also need to be addressed.

Each of the aforementioned principles of information security is analyzed in context to the service provided by the application for mobile Android devices that integrates blockchain timestamping and digital signatures based on ICP-Brasil standards.

Andress (2016, p. 29) refers to confidentiality as the ability to protect “data from those who are not authorized to view it”. In the context of the application prototype, confidentiality is provided by signing a document using an ICP-Brasil digital certificate, generating a message digest of the signed document and timestamping the message digest in the blockchain.

A message digest results from calculating the SHA 256 algorithm of the document, and it is a condensed representation of the document (FIPS PUB 180-4, 2018, p. 3), of 64 digits in fixed-size. Additionally, it represents the “DNA” of the document (ORIGINALMY, 2020).

Andress (2016, p. 29) refers to integrity as preventing “data from being changed in an unauthorized or undesirable manner”. In the context of the application prototype, a SHA 256 algorithm is used for proof of integrity since “a change to any bit or bits in M results, with high probability, in a change to the hash code” (STALLINGS, 2014, p. 314).

Through the SHA 256 algorithm, in order to prove the integrity of a document, a message digest of a document signed with an ICP-Brasil digital certificate is recalculated using the SHA 256 algorithm and compared to the message digest that

was timestamped in the blockchain to detect whether the document was changed (FIPS PUB 180-4, 2018, p. iii).

Whitman and Mattord (2016, p. 14) refer to authenticity as “an attribute of information that describes how data is genuine or original rather than reproduced or fabricated.” In the context of the application prototype, there is a two-step process to prove the authenticity of a document.

The first step verifies the integrity of a document, which is described in section 2.7.2: a message digest of a document signed with an ICP-Brasil digital certificate is recalculated using SHA 256 algorithm and compared to the message digest that was timestamped in the blockchain in order to detect whether the document has been changed (FIPS PUB 180-4, 2018, p. iii).

The second step verifies the identity of the person or entity that signed the document through implementation of a digital signature validator defined by the ICP-Brasil Standards (DEMOISELLE SIGNER, 2019).

This two-step process proves the authenticity of a document both in terms of data integrity and identity of the digital signature (STALLINGS, 2014, p. 260).

Andress (2016, p. 29) refers to availability as the ability to access “data when we need it”. In the context of the application prototype, the user must be online in order to check for certificate revocation and timestamp a signed document in the blockchain.

Prior to signing a document, every user of the application prototype must be online to check if the digital certificate has been added to the ICP-Brasil Certificate Revocation List (CRL), which registers digital certificates that need to be revoked when the user’s private-key is compromised in case, for example, the user’s computer gets hacked or stolen (FERGUSON; SCHNEIER, 2003, p. 334).

Additionally, the user of the application prototype must be online to timestamp a signed document in the blockchain since the bitcoin system “is structured as a peer-to-peer network architecture on top of the Internet” (ANTONOPOULOS, 2014, p. 139).

Andress (2016, p. 24) refers to identity as “simply an assertion of who we are. This may include who we claim to be as a person, who a computer system claims to be over the network, who the originating party of an e-mail claims to be, what authority we claim to have, or similar transactions”.

In the context of the application prototype, an ICP-Brasil certificate is equivalent to a digital identity. This enables unambiguous identification of a natural person or legal entity that signs a digital document (SAFEWEB, 2019).

In regards to the principle of legality, a document signed with an ICP-Brasil certificate has legal validity equivalent to a document signed on paper (ITI, 2020b). Additionally, any form of electronic document admitted by the parties recognizing a document also has legal validity equivalent to a document signed on paper according to paragraph 2 of Provisional Measure 2.200-2/2001:

Paragraph 2. The provisions of this Provisional Measure do not preclude the use of any other means of proving the authorship and integrity of documents in electronic form, including those that use certificates not issued by ICP-Brasil, as long as admitted by the parties as valid or accepted (BRASIL. Medida Provisória, 2001).

In this regard, a proof of authenticity service provided through the use of blockchain timestamping and digital signatures based on ICP-Brasil Standards has legal validity equivalent to a document signed on paper.

Stallings (2011, p. 19) refers to the principle of non-repudiation as preventing

either sender or receiver from denying a transmitted message. Thus, when a message is sent, the receiver can prove that the alleged sender in fact sent the message. Similarly, when a message is received, the sender can prove that the alleged receiver in fact received the message (STALLINGS, 2011, p. 19).

In the context of the application prototype, a user cannot deny having signed the document using an ICP-Brasil certificate because every user goes through a process of identifying itself in person to a Registration Authority (RA), assuring that the user owns the digital certificate (FERGUSON; SCHNEIER, 2003, p. 29).

Bashir (2017, p. 23) refers to the principle of immutability as

key feature of blockchain: records once added onto the blockchain are immutable. There is the possibility of rolling back the changes but this is considered almost impossible to do as it will require an unaffordable amount of computing resources. For example, in much desirable case of bitcoin if a malicious user wants to alter the previous blocks then it would require computing the PoW (Proof of Work) again for all those blocks that have already been added to the blockchain. This difficulty makes the records on a blockchain practically immutable (BASHIR, 2017, p. 23).

Even though, the ICP-Brasil standard has regulated a Timestamp Authority (TA) protocol (ITI, 2020d), it does not provide the principle of immutability of peer-to-peer network of the decentralized bitcoin system.

In the context of the application prototype, immutability is provided to a document signed with an ICP-Brasil digital certificate that is timestamped in the blockchain “due the existence of a long chain of blocks makes the blockchain’s deep history immutable, a key feature of bitcoin’s security” (ANTONOPOULOS, 2014, p. 164).

2.8 SMART CONTRACTS

According to Singh *et al.* (2018, p.1) smart contracts:

are self-executing contracts where users can codify their agreements and trust relations, which are then stored on a hosting blockchain. Smart contracts can facilitate safe and trusted business activities by providing automated transactions without the supervision of an external financial system such as banks, courts, or notaries. These transactions are traceable, transparent, and irreversible (SINGH, 2018, p. 1).

Furthermore, “smart contracts refer to binding contracts between two or more parties and are enforced in a decentralized manner by the blockchain without the need for a centralized enforcer” (KARAME; ANDROULAKI, 2016, p. 172). Therefore, a smart contract’s “platform is decentralized, which means there is no possible single point of failure. Hence, all the apps will always stay online and never switch off” (COINTELEGRAPH. What is Ethereum, 2020). Smart contracts can “be used to control the ownership of properties. These properties might be tangible (e.g., houses, automobiles) or intangible (e.g., shares, access rights)” (NOFER, M. *et al.*, 2017, p. 185).

The bitcoin cryptocurrency system allows for constrained usage of smart contracts because of its “limited set of variables, transaction types, and data storage [that] seemed to limit the types of applications that could run on top, as second layer solutions” (ANTONOPOULOS; WOOD, 2018, p. 18).

In the bitcoin system, multisignature transactions are used to construct smart contracts in order to achieve the following types of contracts (KARAME; ANDROULAKI, 2016, p. 172):

- Making a Deposit:

there are a number of application scenarios where users need to make deposits (e.g., when using a service which requires assurance in case of damage or misuse). Bitcoin can plan for this case by enabling the creation of deposits to potentially untrusted entities (KARAME; ANDROULAKI, 2016, p. 172).

- Dispute Mediation:

the aforementioned process of making deposits can be inherently extended to deal with dispute mediation. For instance, A and B can agree on a neutral dispute mediator M. Here, all transactions issued by A can be constructed so that they can be spent using the signatures of any two out of the three parties: A, B, and M (KARAME; ANDROULAKI, 2016, p. 172).

- Managing Multiuser Funds:

bitcoin additionally enables different users to collaboratively raise funds for any given project without the need for an external arbiter, for example, to handle disputes. For instance, assume that different entities A_1, \dots, A_n decide to collaboratively raise funds of v BTCs in order to support a project. In this case, it is required that if v BTCs could not be jointly raised, then the funds committed by each entity should be reimbursed (KARAME; ANDROULAKI, 2016, p. 172).

Even though, this research project's application prototype uses blockchain to timestamp document signed with an ICP-Brasil digital certificate, the prototype does not use smart contracts since its application is not deployed in the blockchain. Instead, it is deployed on a Linux-based Virtual Machine (VM) through an implementation of the Bitcoin protocol developed in the Java programming language, called bitcoinJ (BITCOINJ, 2017).

2.9 RELATED WORKS

ICP-Brasil's CA Valid Certificadora provides a service called VALIDChain that uses smart contracts deployed on the Ethereum cryptocurrency to allow "blockchain users to use timestamps to perform digital signatures according to the standards defined by ICP-Brasil" (VALID CERTIFICADORA BLOG, 2020). The Ethereum cryptocurrency "is a Turing-complete [smart] contract processing and

execution platform based on a blockchain ledger. It is not a clone of bitcoin, but a completely independent design and implementation. Ethereum has a built-in currency, called ether, which is required in order to pay for [smart] contract execution” (ANTONOPOULOS, 2014, p. 232).

The company OriginalMy utilizes blockchain IDs as digital identities to sign and timestamp documents in the blockchain. A blockchain ID is purely a bitcoin address that can send/receive coins. Additionally, it is used to timestamp documents in the blockchain. A blockchain ID or bitcoin address is formed by a pair of related keys – a public-key and a private-key (secret-key) – of a public-key or asymmetric-key encryption (section 2.4) method. The private-key is kept private and the public-key is an address consisted “of a string of letters and numbers [...]. Just like you ask others to send an email to your email address, you would ask others to send you bitcoin to your bitcoin address” (ANTONOPOULOS, 2014, p. 18). (STINSON, 1995, p. 114). Identity association to a blockchain ID is done through a Know Your Customer (KYC) process of verifying a natural person’s national ID or passport.

Another related service that integrates blockchain timestamping and digital identities is called Blockcerts (FIGURE 9), which “is an open standard for building apps that issue and verify blockchain-based official records. These may include certificates for civic records, academic credentials, professional licenses, workforce development, and more” (BLOCKCERTS, 2020a).

Like OriginalMy’s blockchain ID, Blockcerts uses bitcoin addresses for identity association. But Blockcerts allows flexible design of identity association since:

separation of identity is desirable from an architectural layering perspective. For a certification system, it’s reasonable that adopters will want to establish identity in different ways, and we want to give them this flexibility. At the same time, our design doesn’t preclude identity association. Since the Bitcoin addresses can be any address, recipients and issuers can choose ones associated with a curated profile (BLOCKCERTS, 2020b).

Thus, identity association designs such as OriginalMy’s blockchain IDs or ICP-Brasil’s digital certificates can be applied to bitcoin addresses in Blockcerts open standard.

This research project is intended to solve the problem of increased demand to digitalize the notarial system in Brazil by contributing with development of tools

that can facilitate authentication of documents in digital forms, as a result, it proposes integration of blockchain timestamping and digital signatures based on ICP-Brasil digital certificates to authenticate digital documents. This research project's application prototype utilizes a different combination of technologies to integrate blockchain and digital identities in comparison with the related works presented above (TABLE 3).

TABLE 3 – RELATED WORKS FEATURES

	PROTOTYPE	VALIDCHAIN	ORIGINALMY	BLOCKCERTS
IDENTITY ASSOCIATION	ICP-Brasil	ICP-Brasil	Blochain ID	Flexible design
CERTIFICATE REVOCATION LIST (CRL)	ICP-Brasil	ICP-Brasil	-	Flexible design
BLOCKCHAIN TIMESTAMP PERFORMANCE	Bitcoin	Ethereum	Bitcoin, Ethereum Classic	Bitcoin

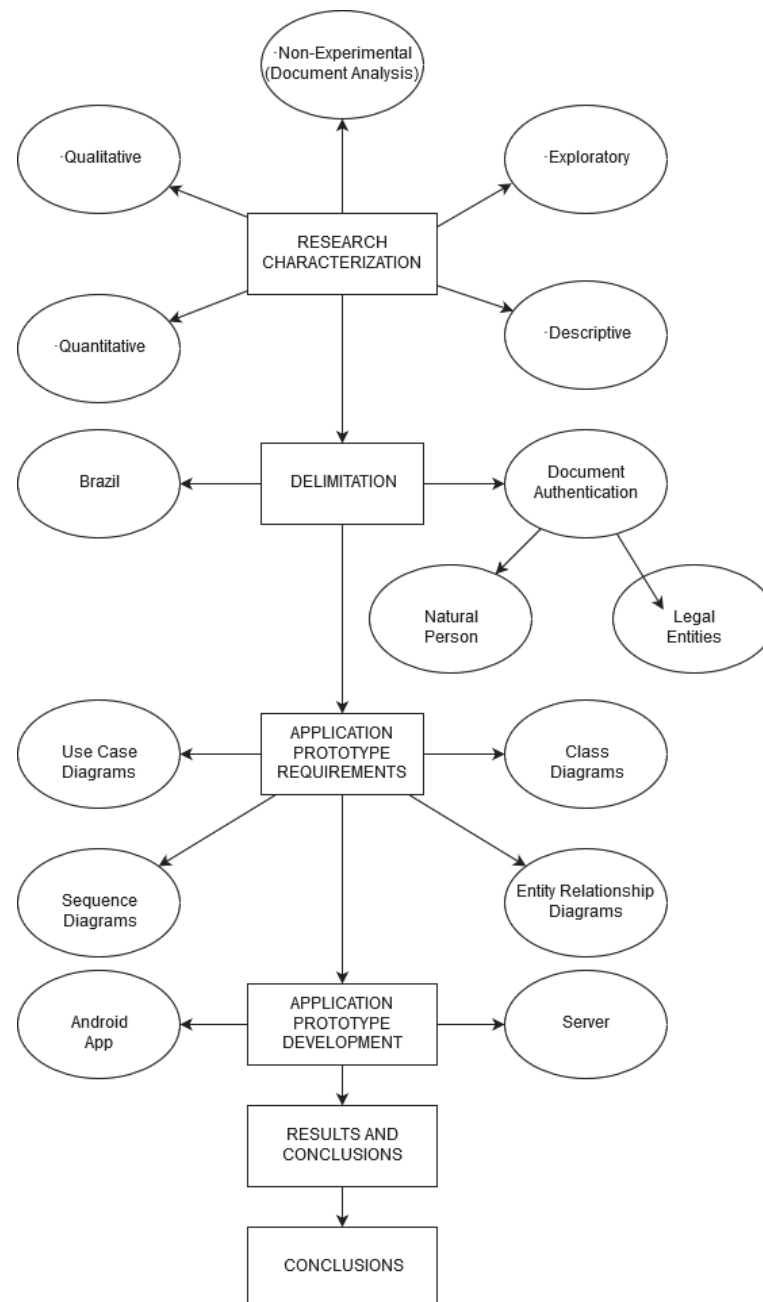
SOURCE: AUTHOR (2020).

This research's application prototype utilizes ICP-Brasil for identity association and certificate revocation list (CRL), and it utilizes bitcoin blockchain to timestamp documents. For development of the application prototype, ICP-Brasil digital identities and CRL were utilized because it is a standard overseen by Brazilian federal government's Instituto Nacional de Tecnologia da Informação (ITI). It provides principles of information security such as identity, legality and non-repudiation (SECTION 2.7 and 5.1). And bitcoin blockchain was utilized for development of the application prototype because it is the most secure network in terms providing principles of authentication, availability, confidentiality, integrity, and immutability (SECTION 2.7) since bitcoin generates the highest hashrate for double-spending prevention through proof of work consensus (SECTION 2.1 and 5.1). In SECTION 4, a comparative analysis between this research project's application prototype and its related works is presented.

3 METHODOLOGY

This research aims to develop an application prototype for mobile Android devices that integrates blockchain timestamping and digital signatures based on ICP-Brasil digital certificates to provide a digital document authentication service. The methodology structure of this research is shown in FIGURE 11.

FIGURE 11 – METHODOLOGY MAP



SOURCE: AUTHOR (2019).

The following methodologies are used to explain this research in this chapter: research characterization, delimitation, prototype engineering requirements and prototype development.

3.1 RESEARCH CHARACTERIZATION

This research is defined as exploratory and descriptive. An exploratory research is conducted to investigate the technical-theoretical procedures (JÚNIOR; PEREIRA; FILHO, 2007, p. 80) regarding the integration of blockchain timestamping and digital signatures based on ICP-Brasil digital certificates. And a descriptive research is conducted to search for relationships between elements of (JÚNIOR; PEREIRA, FILHO, 2007, p. 81) blockchain and ICP-Brasil.

Regarding the outline, a non-experimental research is conducted utilizing documental analysis to understand the integration of blockchain timestamping and digital signatures based on ICP-Brasil digital certificates. The documental analysis is based on books and academic articles about bitcoin and blockchain technology, cryptography and information security. Additionally, analysis is based on website platforms of digital certificate and blockchain companies such as Instituto Nacional de Tecnologia da Informação, Valid Certificadora, OriginalMY, and Proof of Existence, and open source code of technologies such as bitcoinJ, Demoiselle Signer and Blockcerts.

As for the nature, a qualitative research is conducted to explore and describe blockchain and ICP-Brasil digital certificates, seeking integration between the technologies. Additionally, it involves collecting statistical data that justifies integrating blockchain and digital certificates from sources such as Instituto Nacional de Tecnologia da Informação, Receita Federal, and online news platforms.

3.2 DELIMITATION

Even though blockchain's peer-to-peer network "has the potential ability to circumvent the current limitations of geographic jurisdictions" (SWAN, 2015, p. 30), this research's geographic delimitation is set on the region of Brazil because the digital certification based on the ICP-Brasil Standards is maintained and audited by the Brazilian federal government's Instituto Nacional de Tecnologia da Informação

(ITI), and is regulated by Provisional Measure 2.200-2/2001 that has initiated the implementation of ICP-Brasil's national system of digital certification (ITI, 2020g).

In the near future, ICP-Brasil's signed documents will also be accepted within countries of the Mercosul bloc. On December 5th, 2019, during the LV summit of heads of Mercosul states (ITI, 2020f), it was signed a mutual recognition agreement for digital signatures within the scope of Mercosul [which] will enable the exchange of electronic documents between governments, companies and citizens of the countries of the bloc" (ITI, 2020a).

3.3 REQUIREMENTS OF APPLICATION PROTOTYPE

Use case diagrams, sequence diagrams, class diagrams and entity-relationship diagrams of the Unified Modeling Language (UML) were used in order to "specify, visualize and document" (OBJECT MANAGEMENT GROUP, 2019) the application prototype.

3.3.1 USE CASE DIAGRAMS

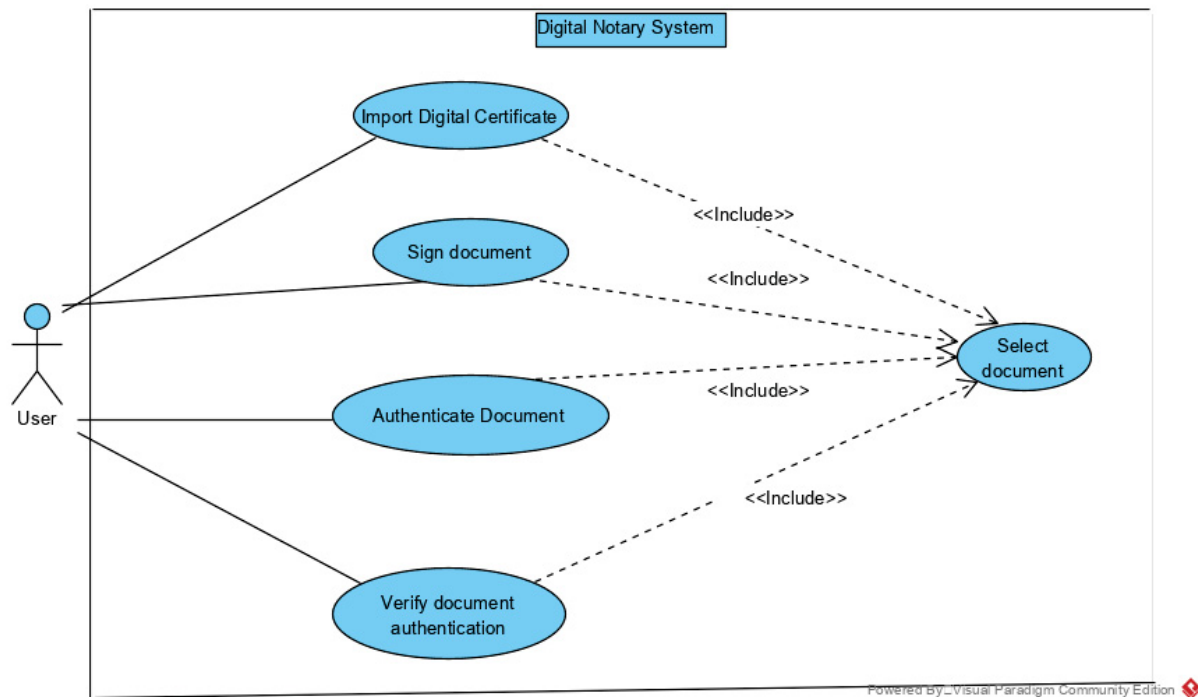
The application prototype has four use cases that are described in FIGURE 12: import digital certificate, document signing, document authentication, and document authentication verification. The user described in FIGURE 12 represents a natural person or legal entity that performs each use case operation in the Android app prototype.

The user communicates with each use case (import digital certificate, sign document, authenticate document, and verify document authentication). According to the UML notation, a solid line connects each interaction between the user and the use cases "sign document", "authenticate document, and verify document authentication" to emphasize a communication between them (FURLAN, 1998, p. 175).

The use case "select document" has a common interaction between the use cases (import digital certificate, sign document, authenticate document, and verify document authentication). In the UML notation, a dotted line with an "include" description connects the use case "select document" to the use cases "sign

document”, “authenticate document, and ”verify document authentication” to emphasize common interaction between them (BEZERRA, 2015, p. 62).

FIGURE 12 – APPLICATION PROTOTYPE USE CASE DIAGRAM



SOURCE: AUTHOR (2019).

3.3.2 PROCESS FLOWCHART

Each use case is an operation in the Android app that is executed following the order of operations described in FIGURE 13.

IMPORT DIGITAL CERTIFICATE:

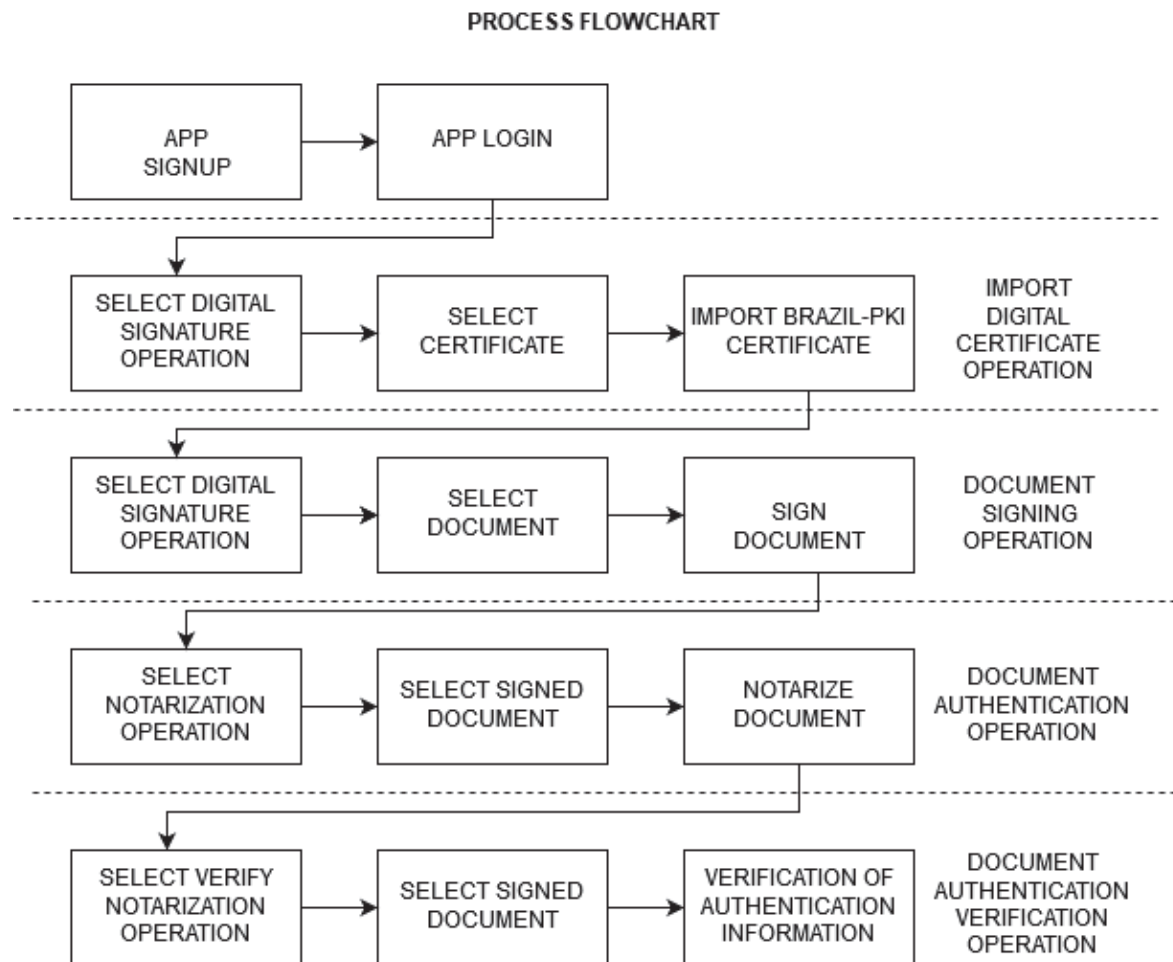
- 1) User selects option to import digital certificate
- 2) System presents option to select a digital certificate to import
- 3) User selects a digital certificate
- 4) System imports digital certificate

DOCUMENT SIGNING:

- 1) User selects option to sign a document
- 2) System presents option to select a document

- 3) User selects a document
- 4) System signs the document

FIGURE 13 – PROCESS FLOWCHART OF APPLICATION PROTOTYPE



SOURCE: AUTHOR (2019).

DOCUMENT AUTHENTICATION:

- 1) User selects option to sign a document
- 2) System presents option to select a document
- 3) User selects a document
- 4) System signs the document

DOCUMENT AUTHENTICATION VERIFICATION:

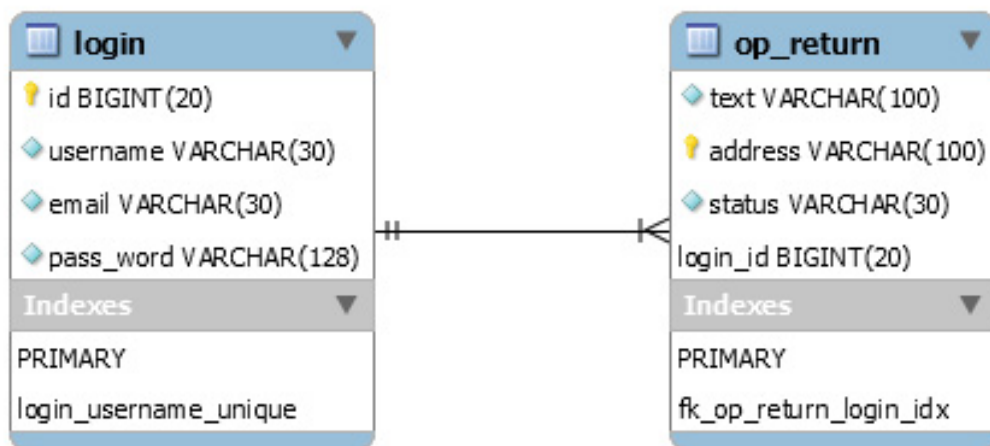
- 1) User selects option to verify document authentication
- 2) User informs "Transaction ID" used to retrieve a document that was timestamped in the blockchain

- 3) System presents option to select a document
- 4) User selects a document
- 5) System shows information of document authentication

3.3.3 DATABASE

The application prototype database has two entities (FIGURE 14): “login” and “op_return”. Entity “login” registers user’s data for login authentication into the application. And entity “op_return” registers data of blockchain timestamps.

FIGURE 14 – APPLICATION PROTOTYPE ENTITY-RELATIONSHIP DIAGRAM



SOURCE: AUTHOR (2019).

The relationship between entity “login” and “op_return” is “One-to-Many” since a user registered in entity “login” can have several records of signed document timestamps in entity “op_return”.

Attributes of entity “login” have the following designations in the system:

- “id”: primary key of entity “login”. It registers a unique integer value for every new user signed up in the system;
- “username”: an identification chosen by the user;
- “email”: user email used to access the app along with a password combination;

- “pass_word”: user password used to access the app along with a user email combination.

Attributes of entity “op_return” have the following designations in the system:

- “login_id”: foreign key that references attribute “id” of entity “login”.
- “address”: registers bitcoin wallet addresses that receive payment fees from users who timestamp documents signed with an ICP-Brasil digital certificate through the app.
- “text”: registers a SHA 256 message digest of a document signed with an ICP-Brasil digital certificate.
- “tx_id”: registers a transaction ID that is used to retrieve a SHA 256 message digest of a document signed with an ICP-Brasil digital certificate in the blockchain. It is used to verify the integrity of the document by comparing it to the SHA 256 message digest of the user’s a document signed with an ICP-Brasil digital certificate to detect whether the document was changed.
- “status”: registers the status of a timestamp process. And it has three status:
 1. Status "INVALID_DATA": refers to attribute “address”, which represents an empty bitcoin address that has not yet received a transaction of a signed document timestamp.
 2. Status "WAITING_TX": refers to attribute “address”, which represents a bitcoin address that has received a transaction of a signed document timestamp, but has not yet been confirmed in the blockchain.
 3. Status “REGISTERED”: refers to attribute “address”, which represents a bitcoin address that has received confirmation of a signed document timestamp transaction in the blockchain.

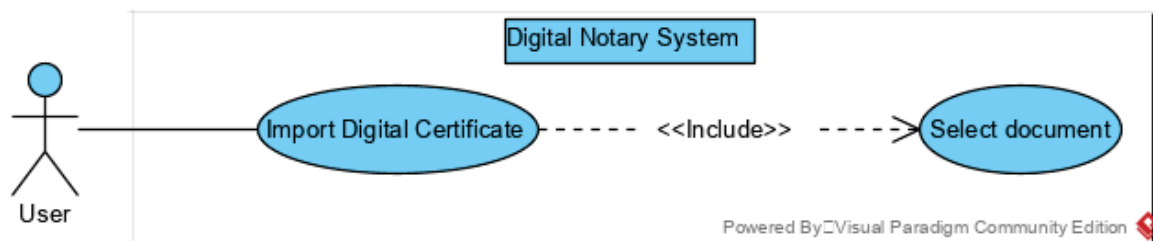
3.3.4 IMPORT DIGITAL CERTIFICATE

Import digital certificate operation is described in next sections through the following UML diagrams: use case diagram, class diagram and sequence diagram.

3.3.4.1 USE CASE DIAGRAM OF IMPORT DIGITAL CERTIFICATE

In the FIGURE 15, the user communicates with the import digital certificate use case through the following steps:

FIGURE 15 – IMPORT DIGITAL CERTIFICATE USE CASE DIAGRAM



SOURCE: AUTHOR (2019).

- 1) User selects option to import digital certificate
- 2) System presents option to select a digital certificate to import
- 3) User selects a digital certificate
- 4) System imports digital certificate

3.3.4.2 CLASS DIAGRAM OF IMPORT DIGITAL CERTIFICATE

The class diagram in FIGURE 16 shows association of classes that are used when importing an ICP-Brasil certificate in the Android app of the application prototype. Classes in this diagram represent a static state of operation “Import Digital Certificate”.

FIGURE 16 – ANDROID APP IMPORT DIGITAL CERTIFICATE CLASS DIAGRAM

SOURCE: AUTHOR (2019)².

Class “MainActivity” is associated directly with all classes in the above diagram. An instance of class “MainActivity” is always active during the lifecycle of the Android app. This class represents the main window interface for interaction with the user on the Android app.

Class “MainActivity” has an association relationship with class “KeyStorePKCS12”. Class “KeyStorePKCS12” executes the “Import Digital

² A full size image of the FIGURE 16 can be accessed in: https://github.com/apekato/serv/blob/master/src/main/resources/diagrams/CLASS_DIAGRAM_ANDROID_APP_IMPORT_DIGITAL_CERTIFICATE.png

Certificate” operation through implementation of Demoiselle Signer (DEMOISELLE SIGNER, 2019), a Java open source code that provides functionalities for generating and validating ICP-Brasil digital signatures. There is connectivity of “one-to-one” from class “MainActivity” to class “KeyStorePKCS12”, i.e., for every “Import Digital Certificate” operation, there is only one call from class “MainActivity” to a method that imports a digital certificate in class “KeyStorePKCS12”.

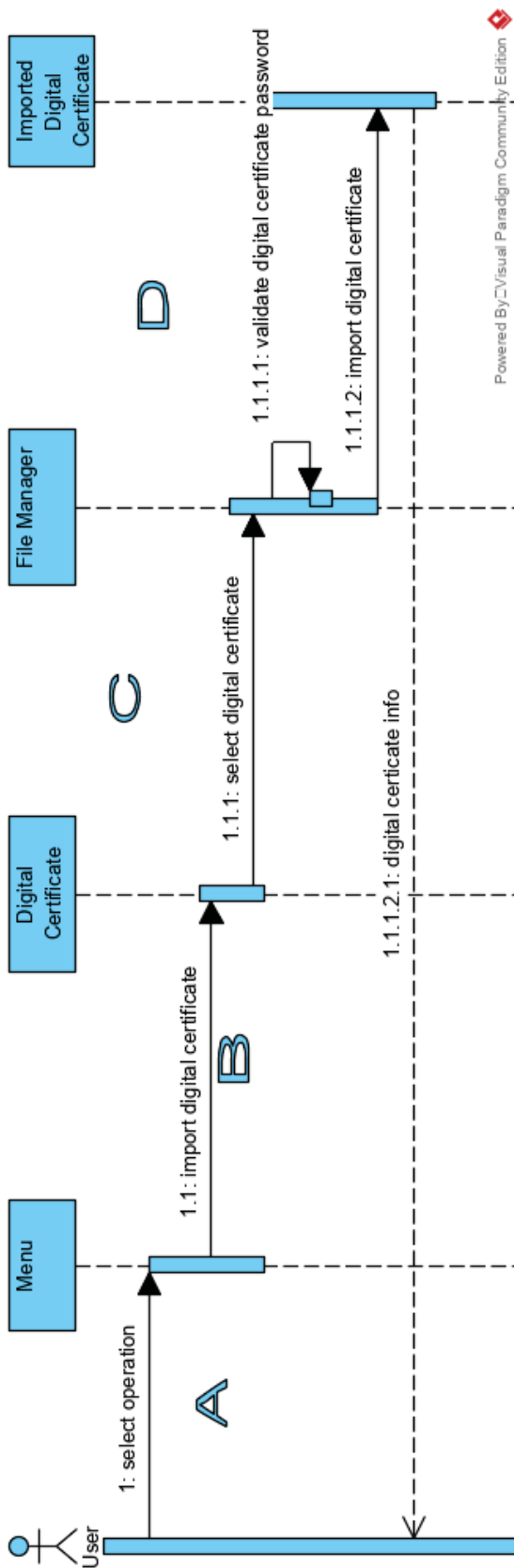
Class “MainActivity” has an association relationship with class “DigCert”. An object of class “DigCert” saves an ICP-Brasil certificate in bytes array format, and saves metadata from the ICP-Brasil certificate, such as filename, name, ID number and certificate expiration date. There is connectivity “one-to-many” from class “MainActivity” to class “DigCert”, i.e., a user can select one ICP-Brasil certificate from a list of one or more ICP-Brasil certificates.

Class “MainActivity” has an association relationship with class “ItemFragment”. An instance of class “ItemFragment” represents a portion of class “MainActivity” window interface. It presents information obtained from an imported ICP-Brasil certificate through an object of class “DigCert”, such as name, ID number and certificate expiration date. There is connectivity “one-to-one” from class “MainActivity” to class “ItemFragment”.

3.3.4.3 SEQUENCE DIAGRAM OF IMPORT DIGITAL CERTIFICATE

The “import digital certificate” operation of the application prototype is described through a sequence diagram. FIGURE 17 presents a sequence diagram that describes the operations to import an ICP-Brasil digital certificate to the Android app. The interactions between the objects are described using screenshots of the prototype's Android app.

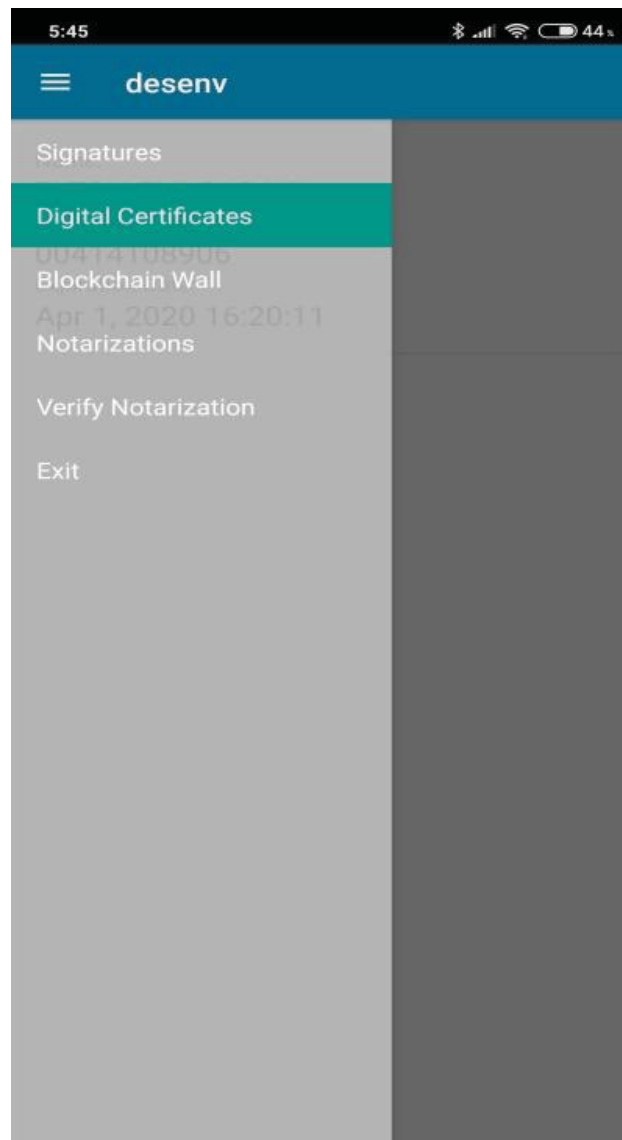
FIGURE 17 – IMPORT DIGITAL CERTIFICATE SEQUENCE DIAGRAM



SOURCE: AUTHOR (2019).

FIGURE 18 illustrates part A (select operation) of the import digital certificate sequence diagram (FIGURE 17). This figure presents a menu of operations to the user, such as “Signatures”, “Digital Certificates”, “Notarizations”, and “Verify Notarization”. To initiate the document signing operation the user taps on “Digital Certificates” on the menu of operations.

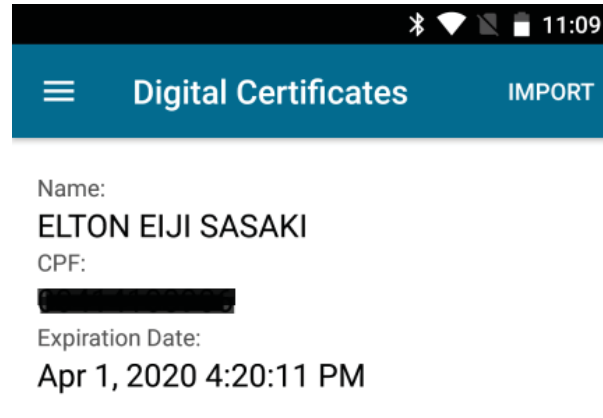
FIGURE 18 – FUNCTIONALITIES MENU ON ANDROID APP



SOURCE: AUTHOR (2019).

FIGURE 19 illustrates part B (import digital certificate) of the import digital certificate sequence diagram (FIGURE17). This figure presents information of an imported ICP-Brasil digital certificate. The “IMPORT” button opens a File Manager to select an ICP-Brasil digital certificate the user wants to import.

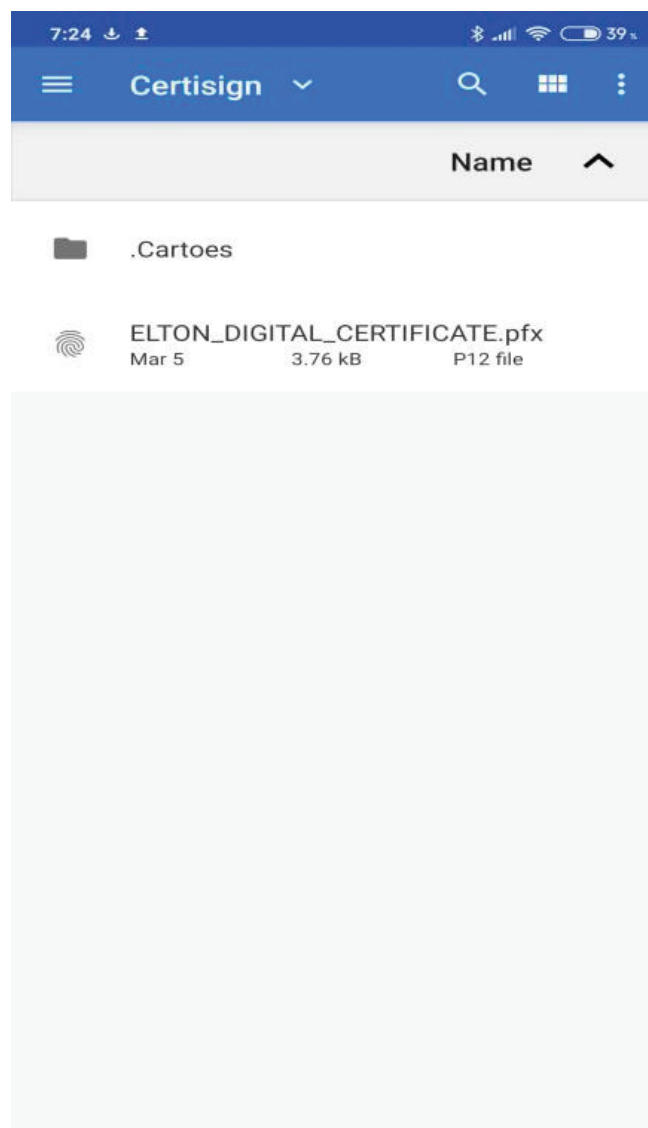
FIGURE 19 – “SIGNATURES” FUNCTIONALITY ON ANDROID APP



SOURCE: AUTHOR (2019).

FIGURE 20 illustrates part C (select digital certificate) of the import digital certificate sequence diagram (FIGURE 17). This figure presents a File Manager to select ICP-Brasil digital certificate the user wants to import. The thumbnails list shows a list of files the user can select for importing an ICP-Brasil digital certificate. By tapping on an ICP-Brasil digital certificate, the certificate is selected and the app automatically opens a message box to validate the user's digital certificate password to import it into the app.

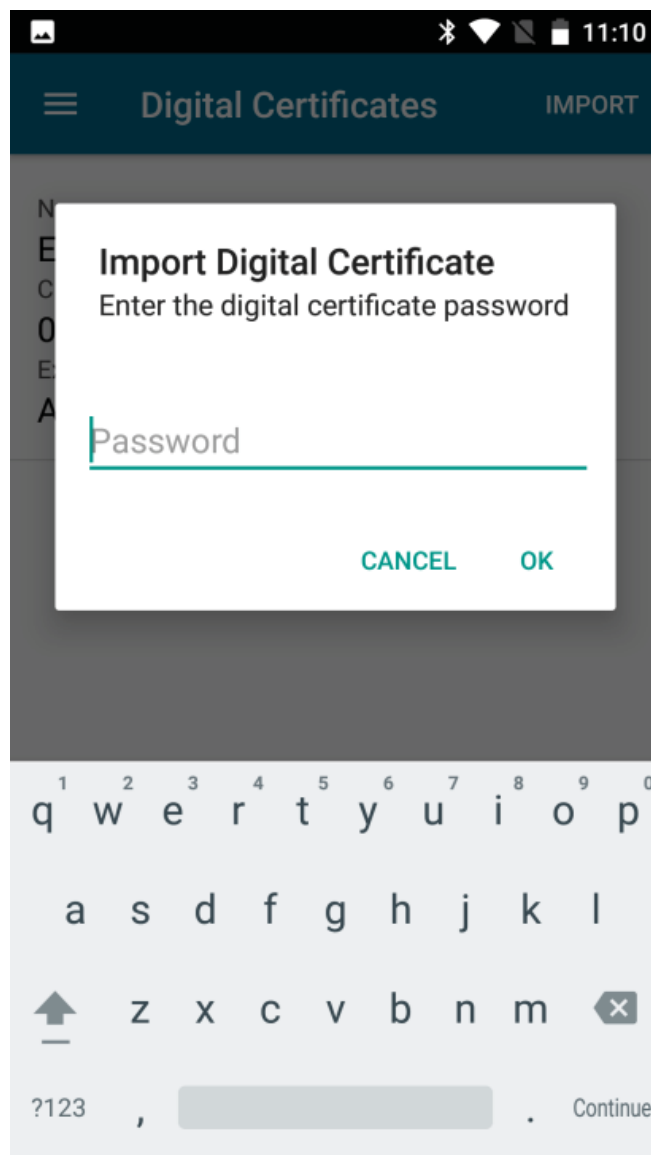
FIGURE 20 – FILE MANAGER TO SELECT CERTIFICATE



SOURCE: AUTHOR (2019).

FIGURE 21 illustrates part D (import digital certificate) of the import digital certificate sequence diagram (FIGURE 17). This figure presents a message box requesting the user to enter the ICP-Brasil certificate password. If the password is authenticated, the certificate is imported into the app and returns to FIGURE 19, presenting information of the imported ICP-Brasil digital certificate. If it isn't authenticated, the user is requested to reenter the ICP-Brasil certificate password.

FIGURE 21 – DIGITAL CERTIFICATE PASSWORD VALIDATOR



SOURCE: AUTHOR (2019).

3.3.5 DOCUMENT SIGNING

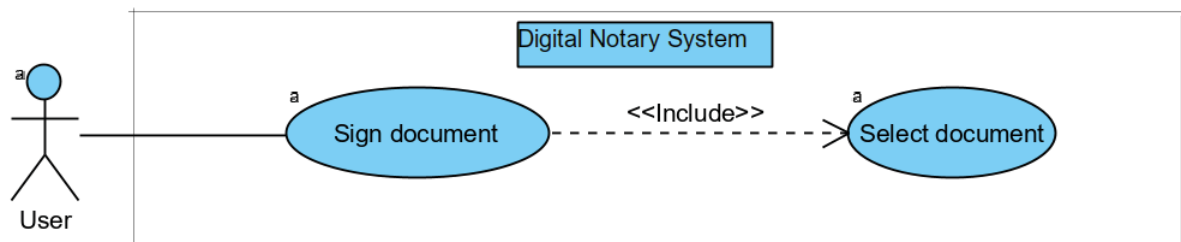
Document signing operation is described in next sections through the following UML diagrams: use case diagram, class diagram and sequence diagram.

3.3.5.1 USE CASE DIAGRAM OF DOCUMENT SIGNING

In FIGURE 22, the user communicates with the document signing use case through the following steps:

- 1) User selects option to sign a document
- 2) System presents option to select a document
- 3) User selects a document
- 4) System signs the document

FIGURE 22 – DOCUMENT SIGNING USE CASE DIAGRAM

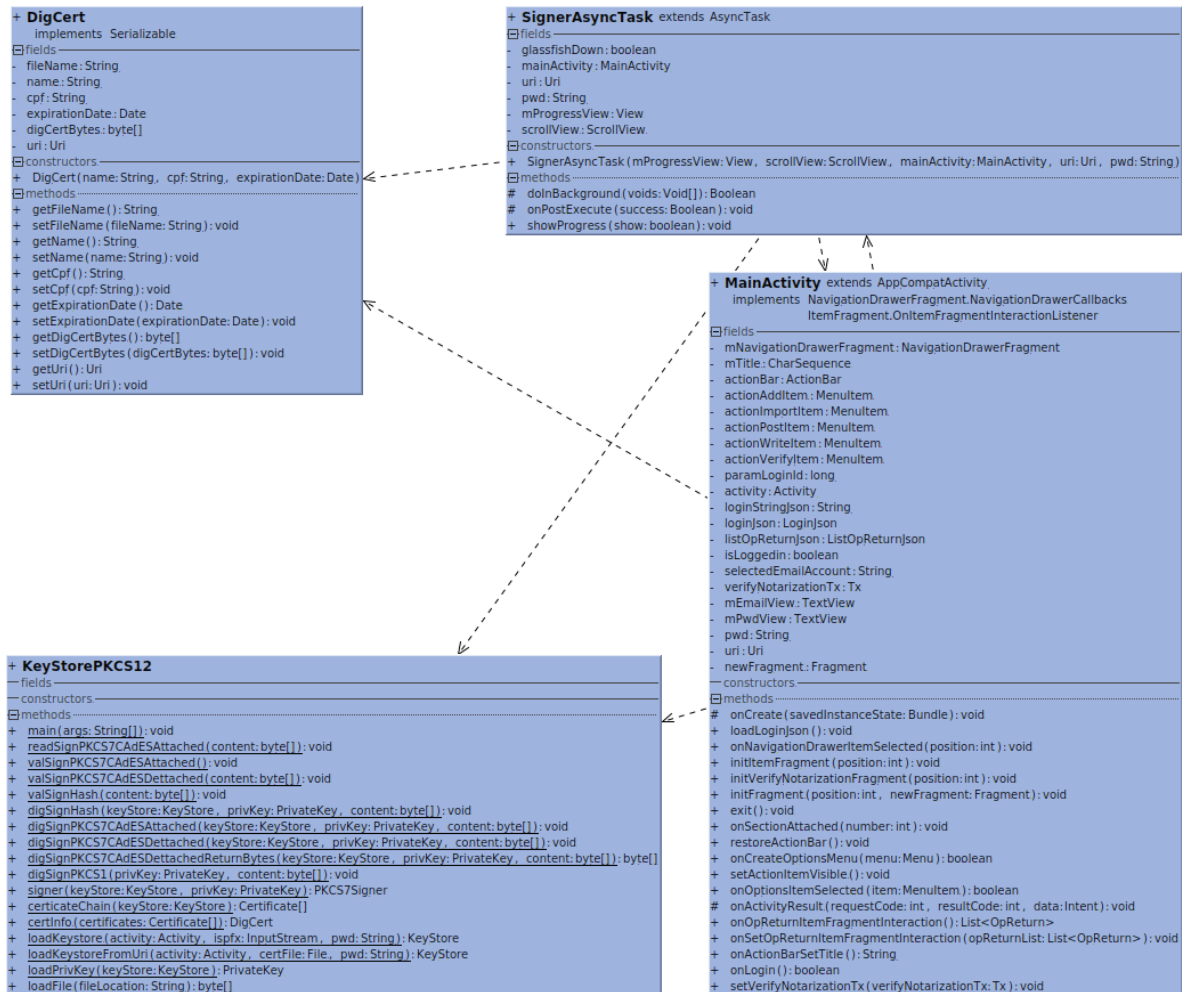


SOURCE: AUTHOR (2019).

3.3.5.2 CLASS DIAGRAM OF DOCUMENT SIGNING

The class diagram in FIGURE 23 shows association of classes that are used when signing a document using an ICP-Brasil certificate.

FIGURE 23 - ANDROID APP DOCUMENT SIGNING CLASS DIAGRAM

SOURCE: AUTHOR (2019)³.

Classes in this diagram represent a static state of a “Document Signing” operation that implements the ICP-Brasil policies of digital signatures.

Class “MainActivity” has an association relationship with class “SignerAsyncTask”. An instance of class “MainActivity” is always active during the lifecycle of the Android app. And it represents the main interface for user interaction with the Android app. There is connectivity “one-to-one” from class “MainActivity” to class “SignerAsyncTask”, i.e., for every “Document Signing” operation, there is one instantiation of class “SignerAsyncTask” from class “MainActivity”.

³ A full size image of FIGURE 23 can be accessed in: https://github.com/apekato/serv/blob/master/src/main/resources/diagrams/CLASS_DIAGRAM_ANDROID_APP_DOCUMENT_SIGNING.png

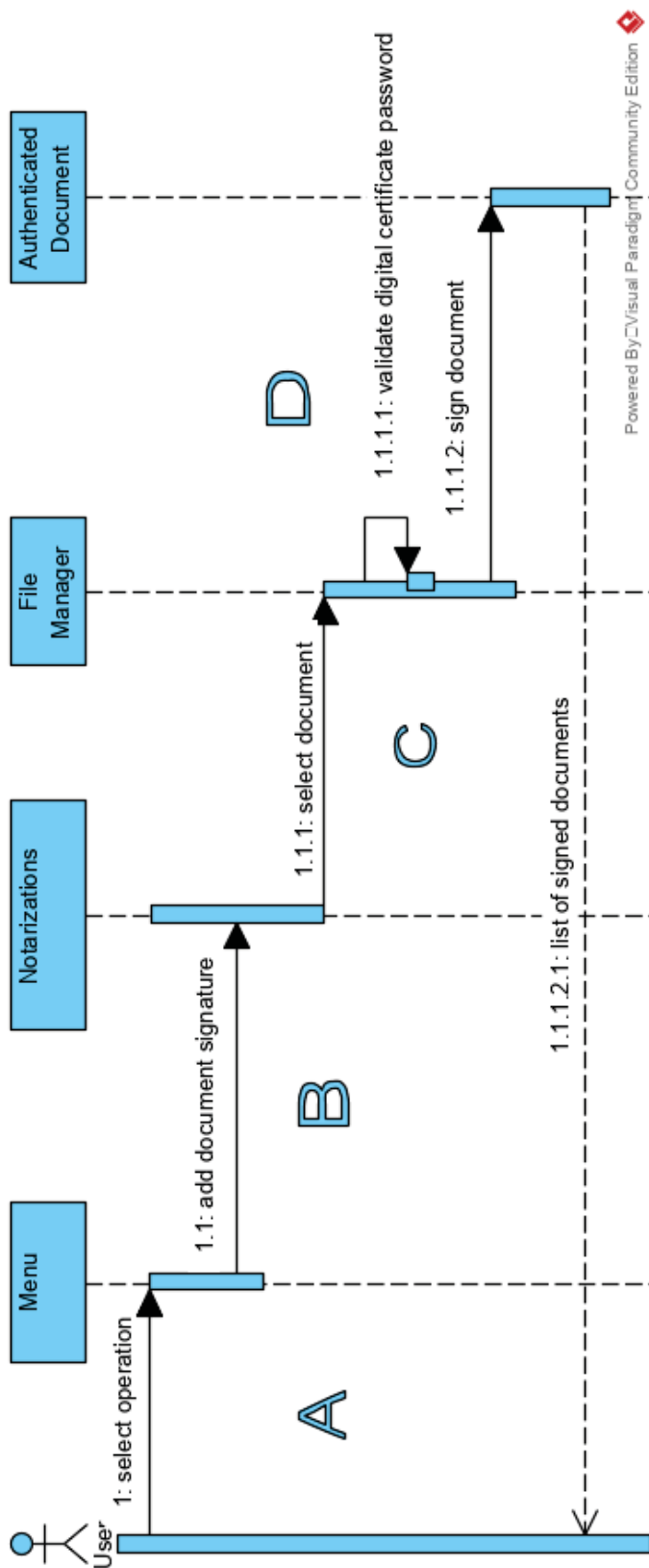
Class “SignerAsyncTask” has an association relationship with class “DigCert”. An object of class “DigCert” saves an ICP-Brasil certificate in bytes array format, and saves metadata from the ICP-Brasil certificate, such as filename, name, ID number and certificate expiration date. Class “DigCert” is used for loading a digital certificate that is internally stored in the Android app, and then it is used for signing a document through class “KeyStorePKCS12”. There is connectivity “one-to-many” from class “SignerAsyncTask” to class “DigCert”, i.e., a user can select one ICP-Brasil certificate from a list of one or more ICP-Brasil certificates.

Class “SignerAsyncTask” has an association relationship with class “KeyStorePKCS12”. Class “KeyStorePKCS12” executes the “Document Signing” operation through implementation of Demoiselle Signer (DEMOISELLE SIGNER, 2019), a Java open source code that provides functionalities for generating and validating ICP-Brasil digital signatures. There is connectivity of “one-to-one” from class “SignerAsyncTask” to class “KeyStorePKCS12”, i.e., for every “Document Signing” operation, there is only one call from class “SignerAsyncTask” to a method that signs a document in class “KeyStorePKCS12”.

3.3.5.3 SEQUENCE DIAGRAM OF DOCUMENT SIGNING

The “Document Signing” operation of the application prototype is described through a sequence diagram. FIGURE 24 presents a sequence diagram that describes the operations to sign a document using an ICP-Brasil certificate. The interactions between the objects are described using screenshots of the prototype's Android app.

FIGURE 24 – DOCUMENT SIGNING SEQUENCE DIAGRAM

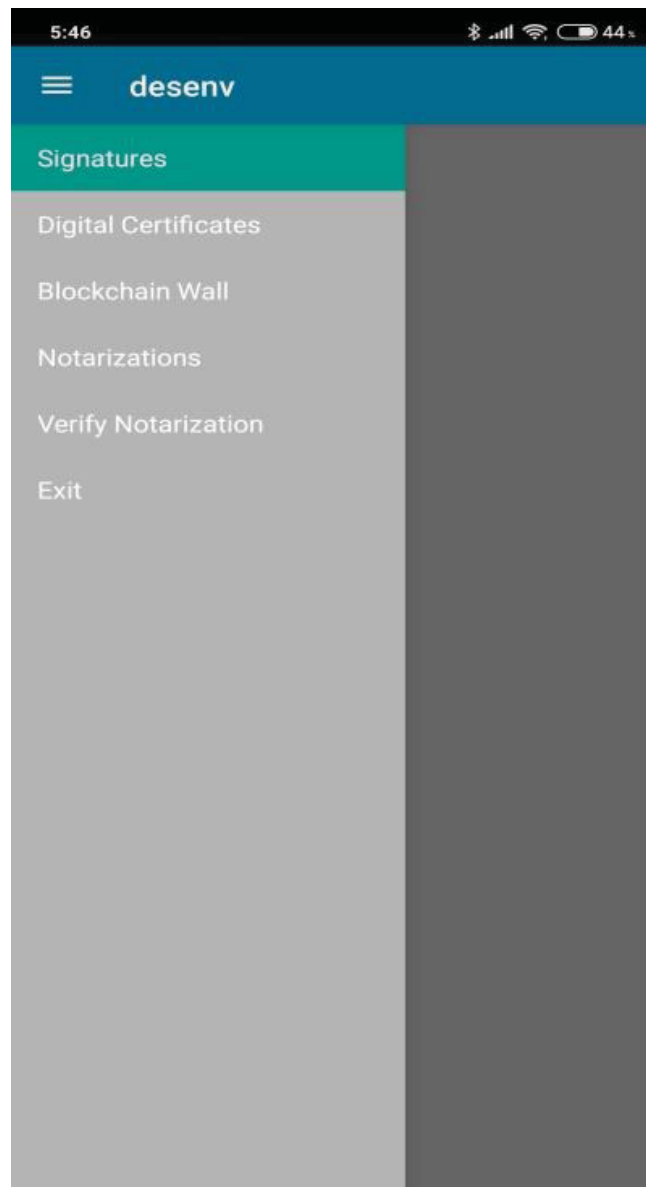


Powered By Visual Paradigm Community Edition

SOURCE: AUTHOR (2019).

FIGURE 25 illustrates part A (select operation) of the “Document Signing” sequence diagram (FIGURE 24). This figure presents a menu of operations to the user, such as “Signatures”, “Digital Certificates”, “Notarizations”, and “Verify Notarization”. To initiate the “Document Signing” operation the user taps on “Signatures” on the menu of operations.

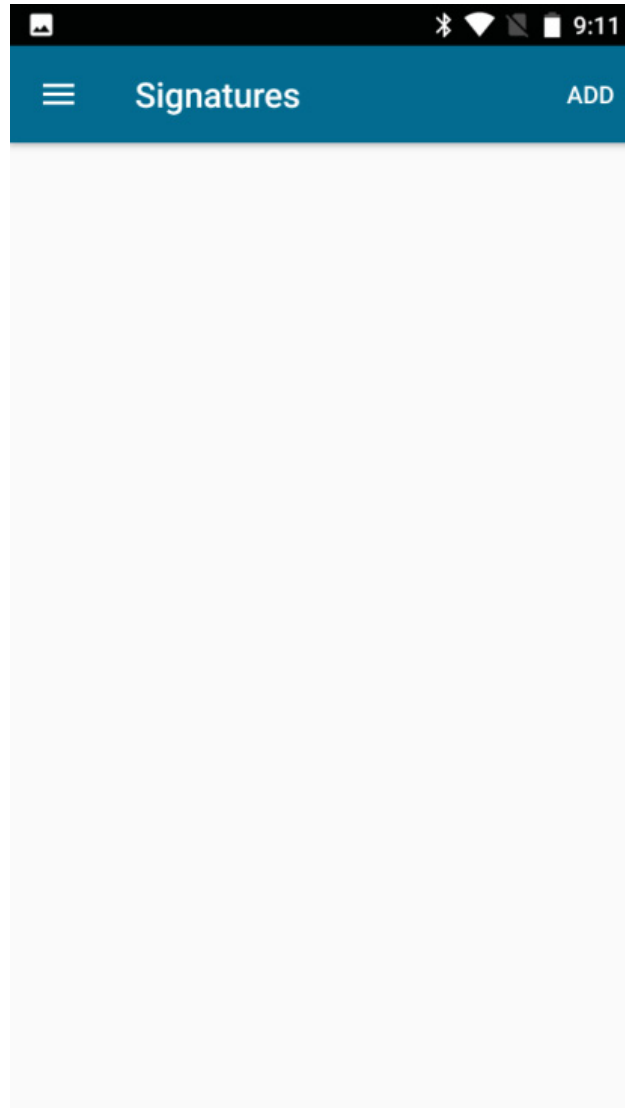
FIGURE 25 – FUNCTIONALITIES MENU ON ANDROID APP



SOURCE: AUTHOR (2019).

FIGURE 26 illustrates part B (add document signature) of the “Document Signing” sequence diagram (FIGURE 24). This figure presents a list of documents signed with an ICP-Brasil digital certificate. The “ADD” button opens a File Manager to select a document the user wants to sign.

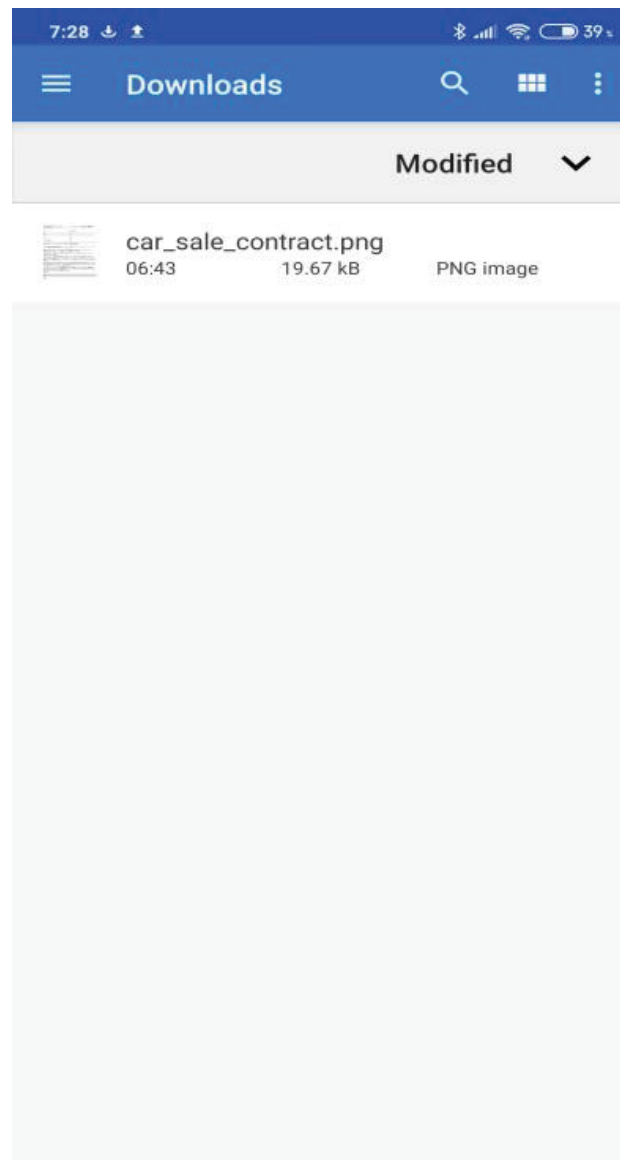
FIGURE 26 – “SIGNATURES” FUNCTIONALITY ON ANDROID APP



SOURCE: AUTHOR (2019).

FIGURE 27 illustrates part C (select document) of the “Document Signing” sequence diagram (FIGURE 24). This figure presents a File Manager to select a document the user wants to sign. The thumbnails list shows a list of documents the user can select for document signing. By tapping on a document, the document is selected and the app automatically opens a message box to validate the user’s digital certificate password to sign the document.

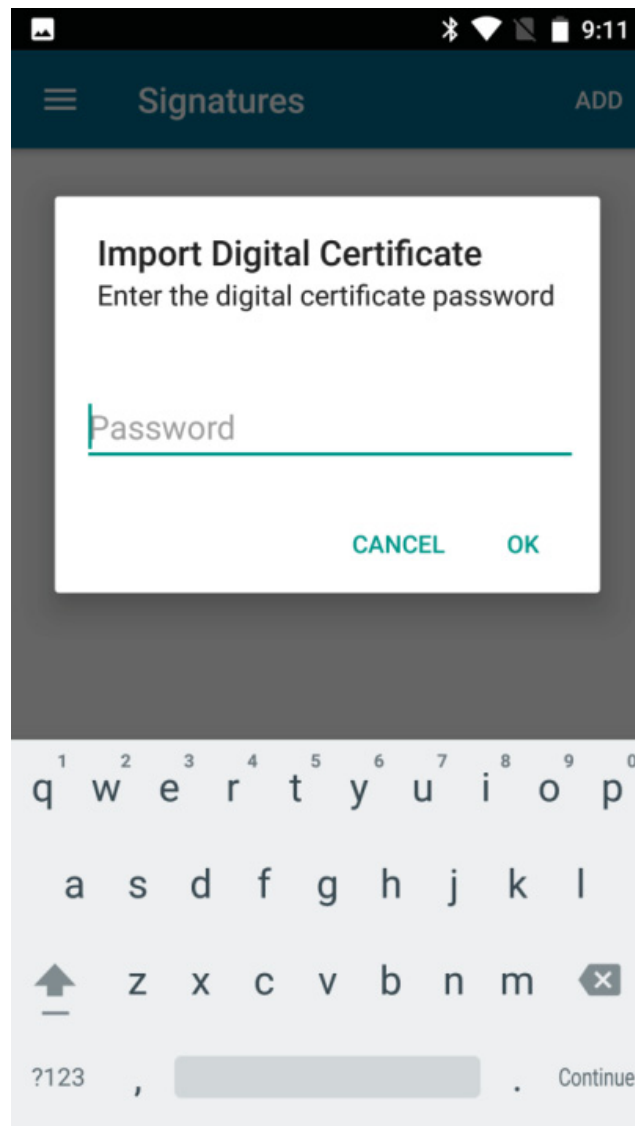
FIGURE 27 – FILE MANAGER TO SELECT SIGNED DOCUMENT



SOURCE: AUTHOR (2019).

FIGURE 28 illustrates part D (sign document) of the “Document Signing” sequence diagram (FIGURE 24). This figure presents a message box requesting the user to enter the ICP-Brasil certificate password. If the password is authenticated, the document gets signed and returns to FIGURE 26, presenting an updated list of documents signed with an ICP-Brasil digital certificate. If it isn’t authenticated, the user is requested to reenter the ICP-Brasil certificate password.

FIGURE 28 – DIGITAL CERTIFICATE PASSWORD VALIDATOR



SOURCE: AUTHOR (2019).

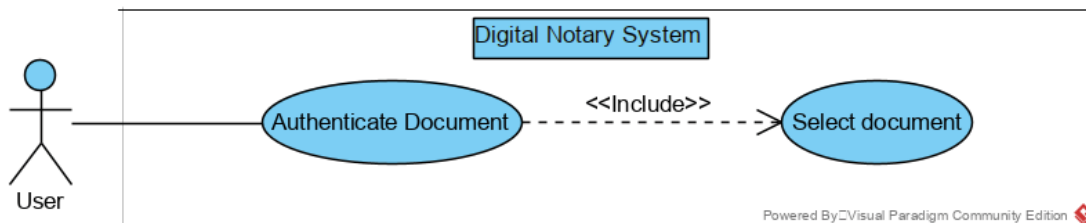
3.3.6 DOCUMENT AUTHENTICATION

“Document Authentication” operation is described in next sections through the following UML diagrams: use case diagram, class diagram and sequence diagram.

3.3.6.1 USE CASE DIAGRAM OF DOCUMENT AUTHENTICATION

In the Figure 29, the user communicates with the document authentication use case through the following steps:

FIGURE 29 – DOCUMENT AUTHENTICATION USE CASE DIAGRAM



SOURCE: AUTHOR (2019).

- 1) User selects option to sign a document
- 2) System presents option to select a document
- 3) User selects a document
- 4) System signs the document

3.3.6.2 CLASS DIAGRAMS OF DOCUMENT AUTHENTICATION

Class diagrams for “Document Authentication” operation is comprised of an Android app class diagram and a server application class diagram.

3.3.6.2.1 CLASS DIAGRAM OF DOCUMENT AUTHENTICATION OPERATION ON ANDROID APP

The class diagram in FIGURE 30 shows association of classes that are used when authenticating a document using an ICP-Brasil certificate on the Android app of

the application prototype. Classes in this diagram represent a static state of operation “Document Authentication”.

Class “MainActivity” is associated directly with all classes in the diagram. An instance of class “MainActivity” is always active during the lifecycle of the Android app. This class is represents the main window interface for interaction with the user on the Android app.

Class “MainActivity” has an association relationship with class “ItemFragment”. An instance of class “ItemFragment” represents a portion of class “MainActivity” window interface. It presents transaction information, retrieved from the server database, of documents that the user authenticated in the blockchain. There is connectivity “one-to-one” from class “MainActivity” to class “ItemFragment”.

FIGURE 30 – ANDROID APP DOCUMENT AUTHENTICATION CLASS DIAGRAM

SOURCE: AUTHOR (2019)⁴.

⁴ A full size image of FIGURE 30 can be accessed in:

https://github.com/apekato/serv/blob/master/src/main/resources/diagrams/CLASS_DIAGRAM_ANDROID_APP_DOCUMENT_AUTHENTICATION.png

Through a “one-to-one” connectivity from class “MainActivity” to class “ItemFragment”, http requests made are using an instance of a web service called “opReturnList” on the prototype application server. In the “Document Authentication” operation, it requests a list of “OpReturn” objects from the application server database.

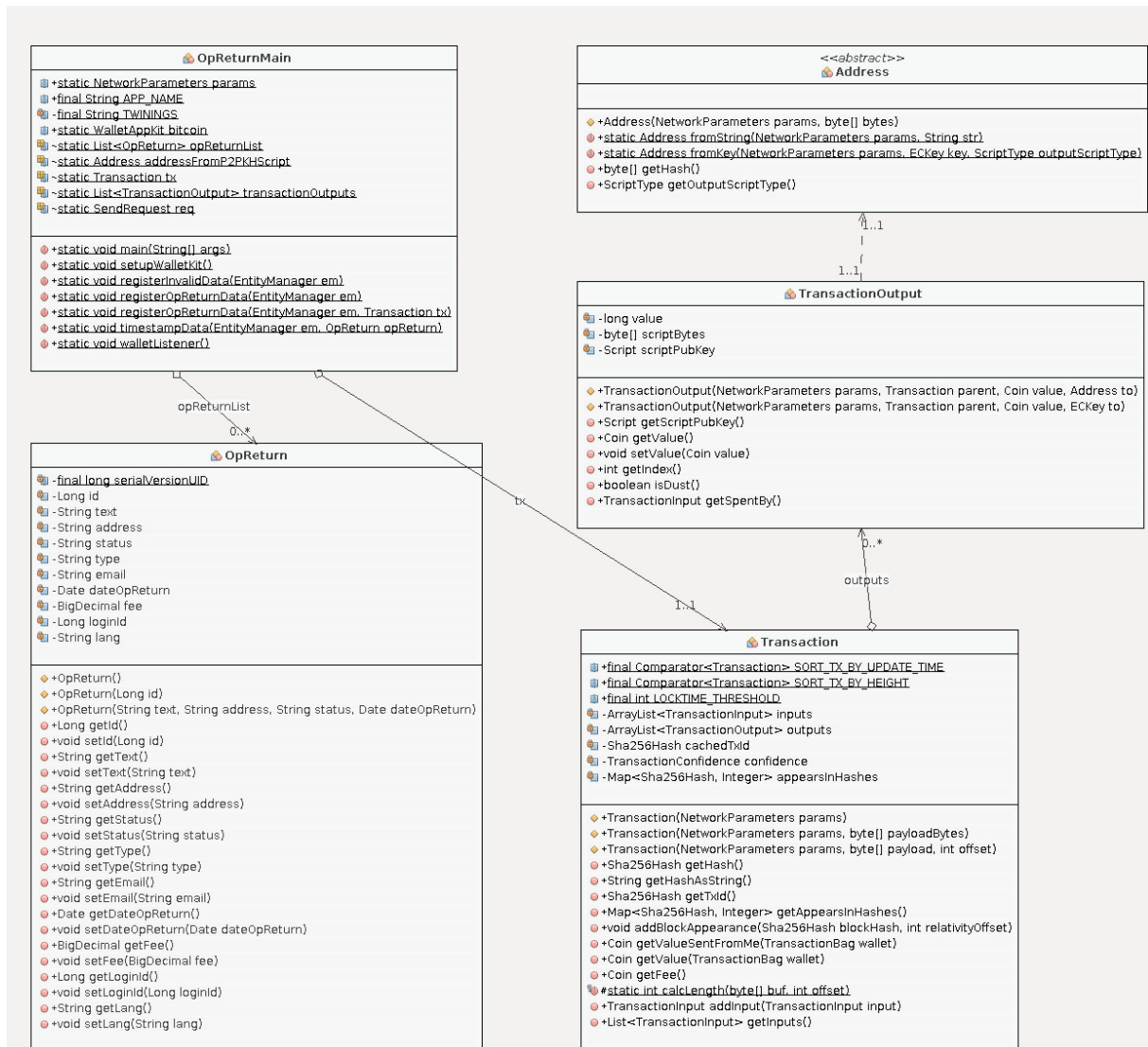
Class “ItemFragment” has an association relationship with class “OpReturn”. An object of class “OpReturn” has the same attributes as entity “op_return” of the application prototype database. The SHA 256 Hash of an authenticated document is stored variable/attribute “text” of an object of class “OpReturn”. The value stored in variable/attribute “text” is presented to the user on the Android app in a list format in order to show document authentications performed by the user. There is connectivity “one-to-many” from class “MainActivity” to class “OpReturn”, i.e., a user can retrieve a list of one or more objects of class “OpReturn”.

Class “MainActivity” has an association relationship with class “OpReturnAsyncTask”. An instance of class “OpReturnAsyncTask” opens a third party bitcoin wallet app installed on Android to initiate a bitcoin fee payment transaction that timestamps a SHA 256 Hash of a document in the blockchain. Additionally, it also loads data of this transaction into an object of class “OpReturn”. This object is saved on entity “op_return” of the application server database through an http request of a web service called “opReturnRequest”. Data saved on entity “op_return” of the application server database keeps track of the status of a timestamp process of a document being authenticated in the blockchain. There is connectivity “one-to-one” from class “MainActivity” to class “OpReturnAsyncTask”.

3.3.6.2.2 CLASS DIAGRAM OF DOCUMENT AUTHENTICATION OPERATION ON SERVER APPLICATION

The class diagram in FIGURE 31 shows association of classes that are used in timestamping data on the bitcoin blockchain. This class diagram is an implementation of a Java open source code of the Bitcoin protocol, called bitcoinJ, which can “maintain a wallet, send/receive transactions without needing a local copy of Bitcoin Core” (BITCOINJ, 2017).

FIGURE 31 – CLASS DIAGRAM OF DOCUMENT AUTHENTICATION

SOURCE: AUTHOR (2019)⁵.

Classes in this diagram represent a static state of a bitcoin transaction that timestamps data in the blockchain. Class “OpReturnMain” is associated directly or indirectly with all classes in the above diagram. This class processes blockchain timestamp transactions by using a script operation code called OP_RETURN that invalidates a transaction output making it unspendable (BITCOIN WIKI, 2019), but “allows a small amount of data to be inserted, which in our case is the document's

⁵ A full size image of FIGURE 31 can be accessed in: https://github.com/apekato/serv/blob/master/src/main/resources/diagrams/CLASS_DIAGRAM_DOCUMENT_AUTHENTICATION.png

[SHA 256] hash” (PROOF OF EXISTENCE, 2019) that is used in timestamping data in the blockchain.

Class “OpReturnMain” has an aggregation association with class “Transaction”, i.e., class “Transaction” is part of class “OpReturnMain”. There is connectivity of “one-to-one” from class “OpReturnMain” to class “Transaction”, i.e., there is only one blockchain timestamp transaction for each instance of class “OpReturnMain”.

Class “Transaction” has an aggregation association with class “TransactionOutput”, i.e., class “TransactionOutput” is part of class “Transaction”. There is connectivity of “one-to-many” from class “Transaction” to class “TransactionOutput”, i.e., a single transaction may be associated with one or several transaction outputs; in other words, a single transaction may send bitcoins to one or more addresses.

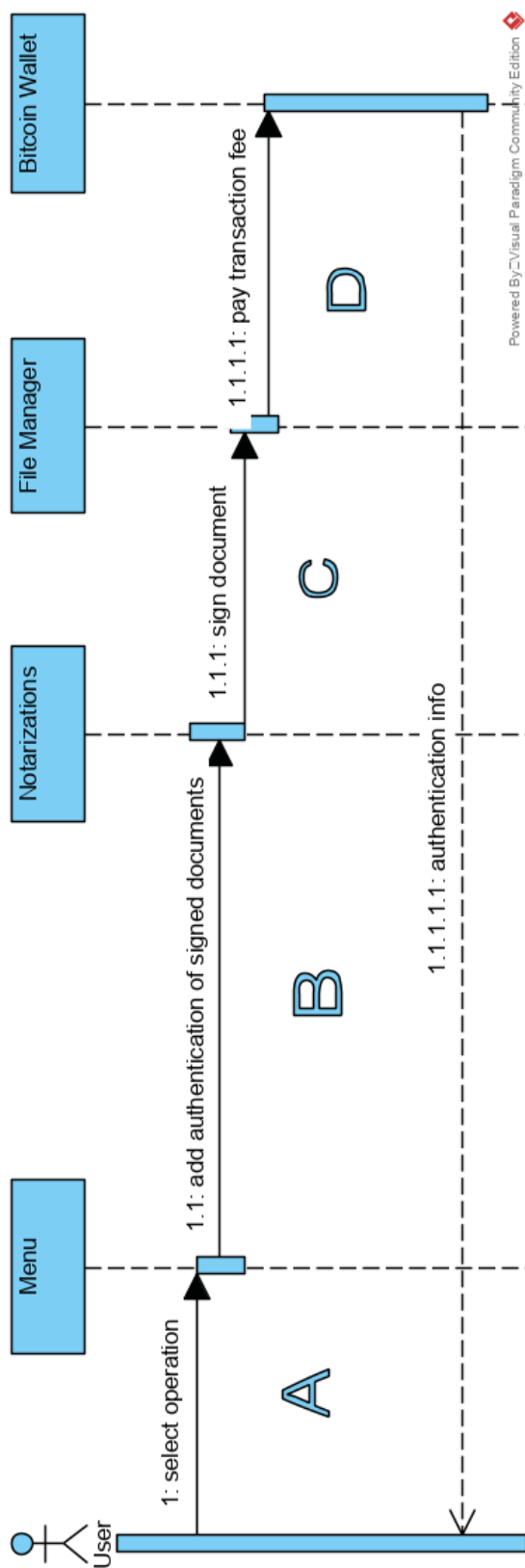
Class “Address” has association of one-to-one with class “Transaction output” since each transaction output is only associated with one bitcoin address.

Class “OpReturn” has an aggregation association with class “OpReturnMain”, i.e., class “OpReturn” is part of class “OpReturnMain”. There is connectivity of “one-to-many” from class “OpReturnMain” to class “OpReturn”. Class “OpReturn” is an object model of database entity “op_return”. It is used persist records of this object on a database.

3.3.6.3 SEQUENCE DIAGRAM OF DOCUMENT AUTHENTICATION

The “Document Authentication” operation of the application prototype is described through a sequence diagram. FIGURE 32 presents a sequence diagram that describes the operations to authenticate a document signed with an ICP-Brasil digital certificate by timestamping it in the blockchain. The interactions between the objects are described using screenshots of the prototype's Android app.

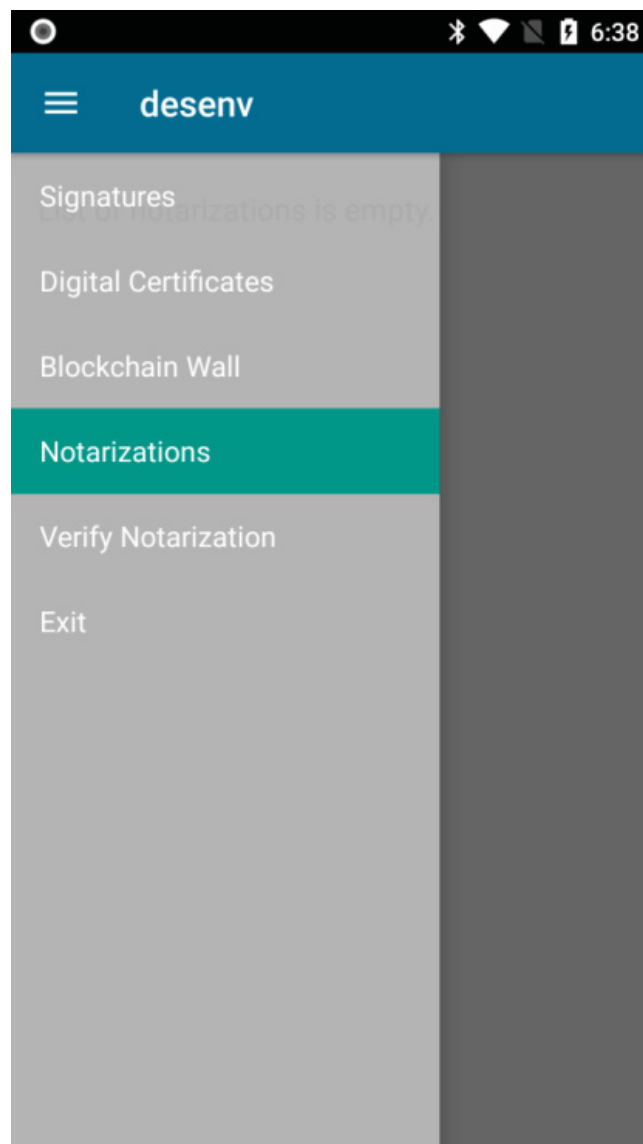
FIGURE 32 – DOCUMENT AUTHENTICATION SEQUENCE DIAGRAM



SOURCE: AUTHOR (2019).

FIGURE 33 illustrates part A (select operation) of the “Document Authentication” sequence diagram (FIGURE 32). This figure presents a menu of operations to the user, such as “Signatures”, “Digital Certificates”, “Notarizations”, and “Verify Notarization”. To initiate the “Document Authentication” operation the user taps on “Notarizations” on the menu of operations.

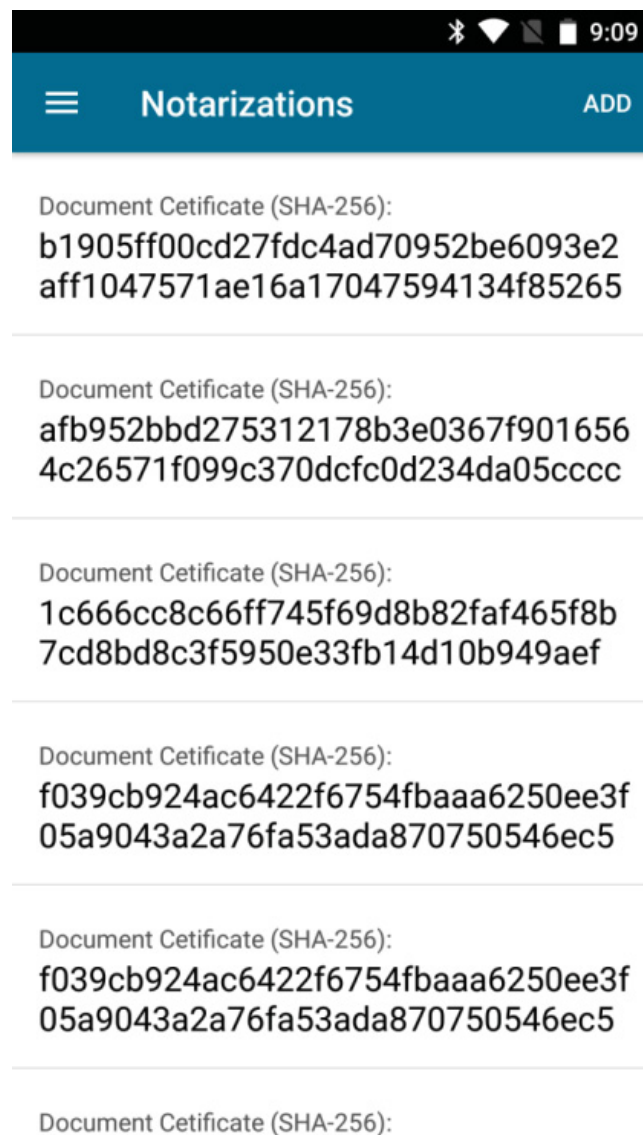
FIGURE 33 – FUNCTIONALITIES MENU ON ANDROID APP



SOURCE: AUTHOR (2019).

FIGURE 34 illustrates part B (add authentication of documents) of the “Document Authentication” sequence diagram (FIGURE 32). This figure presents a list of authenticated documents, i.e., list documents signed with an ICP-Brasil digital certificate timestamped in the blockchain. The “ADD” button opens a File Manager to select a document signed with an ICP-Brasil digital certificate the user wants to authenticate by timestamping it in the blockchain.

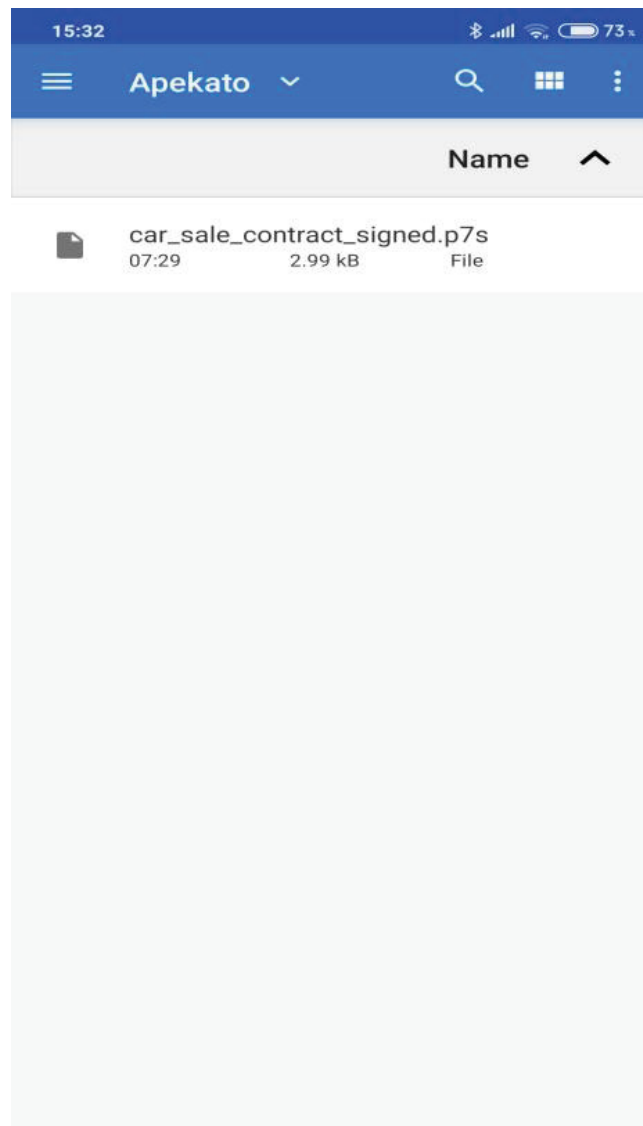
FIGURE 34 – “NOTARIZATIONS” FUNCTIONALITY ON ANDROID APP



SOURCE: AUTHOR (2019).

FIGURE 35 illustrates part C (select document) of the “Document Authentication” sequence diagram (FIGURE 32). This figure presents a File Manager to select a document signed with an ICP-Brasil digital certificate the user wants to authenticate by timestamping it in the blockchain. The thumbnails list shows a list of documents signed with an ICP-Brasil digital certificate the user can select for document authentication. By tapping on a document signed with an ICP-Brasil digital certificate, the document is selected and the app automatically opens a bitcoin wallet.

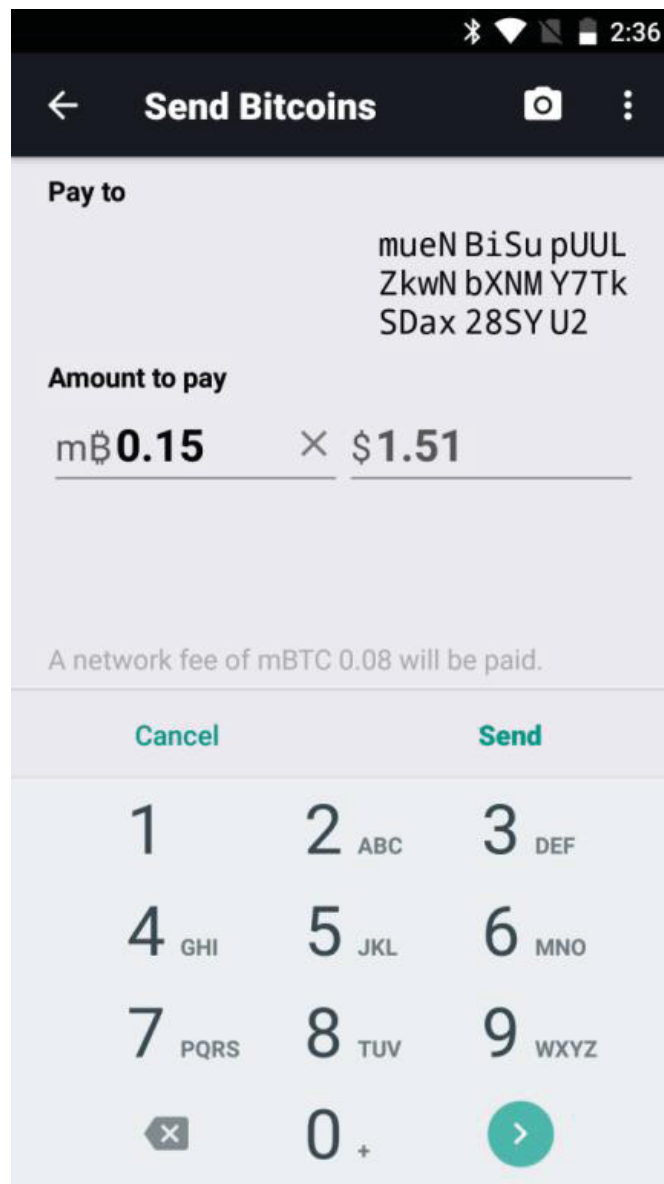
FIGURE 35 – FILE MANAGER TO SELECT SIGNED DOCUMENT FOR NOTARIZATION



SOURCE: AUTHOR (2019).

FIGURE 36 illustrates part D (pay transaction fee) of the “Document Authentication” sequence diagram (FIGURE 32). This figure presents a bitcoin wallet used to pay for a transaction fee to authenticate the document signed with an ICP-Brasil digital certificate with a timestamp in the blockchain. The “Pay to” field indicates the address to send the transaction fee that will process the “Document Authentication”. The “Amount to pay”, informs the cost of the transaction fee of ₧0.00015. By tapping on “Send” button, the user sends the transaction fee to authenticate the signed document with a timestamp in the blockchain.

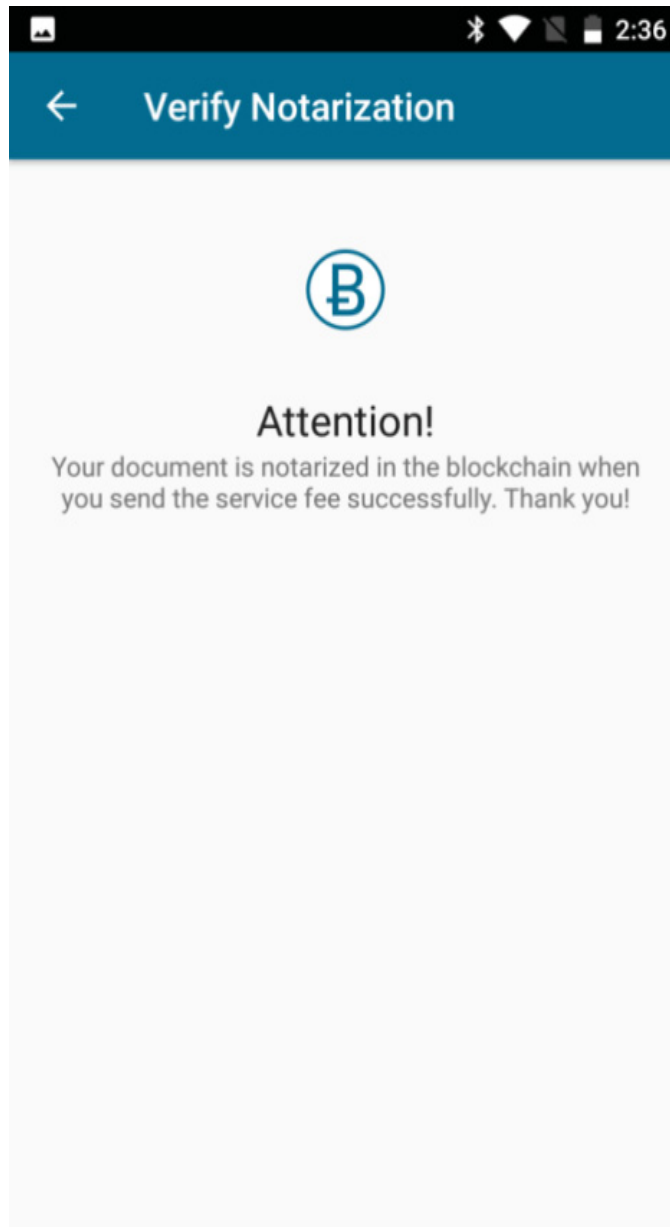
FIGURE 36 – BITCOIN WALLET



SOURCE: AUTHOR (2019).

FIGURE 37 presents a message informing that the document is only authenticated in the blockchain if the bitcoin transaction fee was sent successfully.

FIGURE 37 – SUCCESS AUTHENTICATION MESSAGE



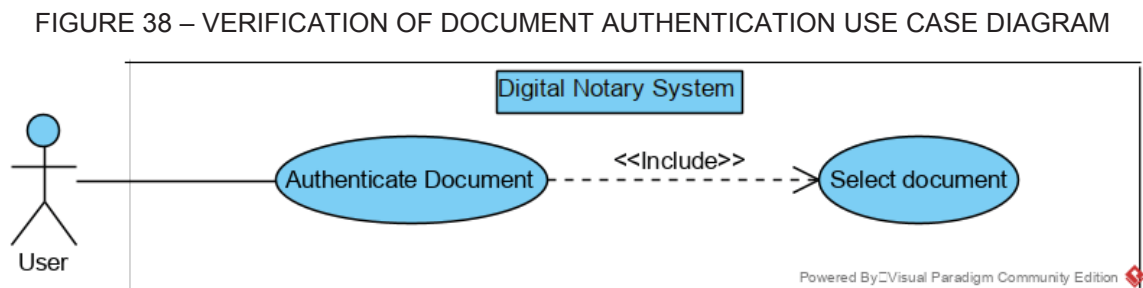
SOURCE: AUTHOR (2019).

3.3.7 VERIFICATION OF DOCUMENT AUTHENTICATION

Verification of “Document Signing” operation is described in next sections through the following UML diagrams: use case diagram, class diagram and sequence diagram.

3.3.7.1 USE CASE DIAGRAM OF VERIFICATION OF DOCUMENT AUTHENTICATION

In FIGURE 38, the user communicates with the verification of document authentication use case through the following steps:



SOURCE: AUTHOR (2019).

- 1) User selects option to verify document authentication
- 2) User informs “Transaction ID” used to retrieve a document that was timestamped in the blockchain
- 3) System presents option to select a document
- 4) User selects a document
- 5) System shows information of document authentication

This figure presents the field “Transaction ID” the user needs to fill out to retrieve the document that was authenticated in the blockchain.

By tapping on “ADD” button, the app retrieves the transaction ID in the blockchain. If the transaction ID is successfully retrieved, it opens a File Manager to select a document the user wants to verify authentication. If it isn’t retrieved, the user is requested to reenter a valid transaction ID.

3.3.7.2 CLASS DIAGRAMS OF VERIFICATION OF DOCUMENT AUTHENTICATION

Class diagrams for “Verification of Document Authentication” operation is comprised of an Android app class diagram and a server application class diagram.

3.3.7.2.1 CLASS DIAGRAM OF VERIFICATION OF DOCUMENT AUTHENTICATION ON ANDROID APP

The class diagram in FIGURE 39 shows association of classes that are used when verifying document authentication on the Android app of the application prototype. Classes in this diagram represent a static state of operation “verification of document authentication”.

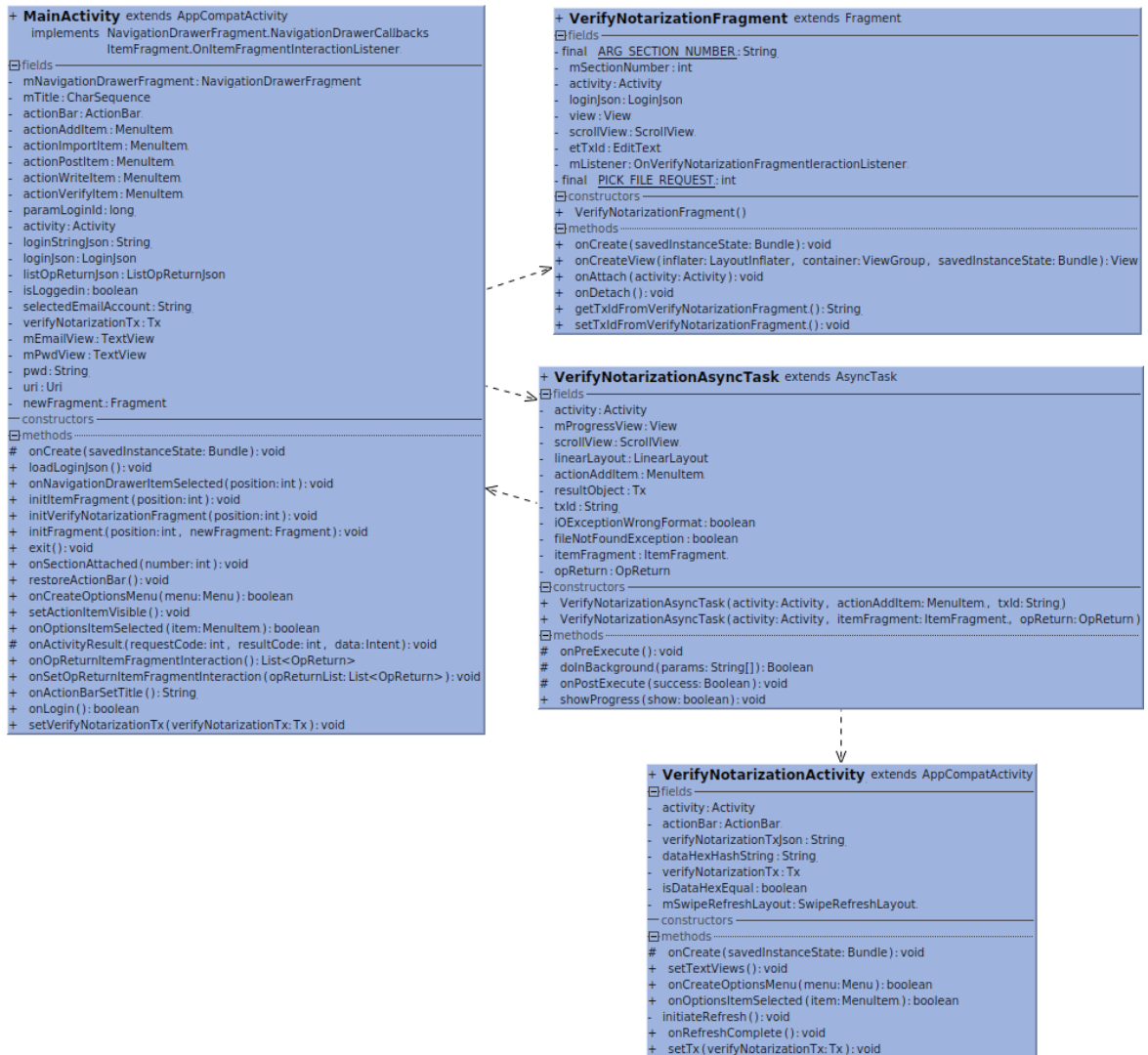
Class “MainActivity” is associated directly with all classes in the diagram. An instance of class “MainActivity” is always active during the lifecycle of the Android app. This class is represents the main window interface for interaction with the user on the Android app.

Class “MainActivity” has an association relationship with class “VerifyNotarizationFragment”. An instance of class “VerifyNotarizationFragment” represents a portion of class “MainActivity” window interface. It presents field called “Transaction ID” where the user pastes a transaction ID of document that was timestamped on the bitcoin blockchain. There is connectivity “one-to-one” from class “MainActivity” to class “VerifyNotarizationFragment”.

Class “MainActivity” has an association relationship with class “VerifyNotarizationAsyncTask”. An instance of class “VerifyNotarizationAsyncTask” is created when the user taps on “ADD” button on the window interface of an instance of class “VerifyNotarizationFragment”. It retrieves the transaction ID in the blockchain. If the transaction ID is successfully retrieved, it opens a File Manager to select a document the user wants to verify authentication. If it isn’t retrieved, the user is requested to reenter a valid transaction ID.

Class “VerifyNotarizationAsyncTask” has an association relationship with class “VerifyNotarizationActivity”. An object of class “VerifyNotarizationActivity” presents a window interface that informs the user if a document signed with an ICP-Brasil digital certificate was timestamped/authenticated in the blockchain.

FIGURE 39 – ANDROID APP VERIFICATION OF DOCUMENT AUTHENTICATION CLASS DIAGRAM



SOURCE: AUTHOR (2019)⁶.

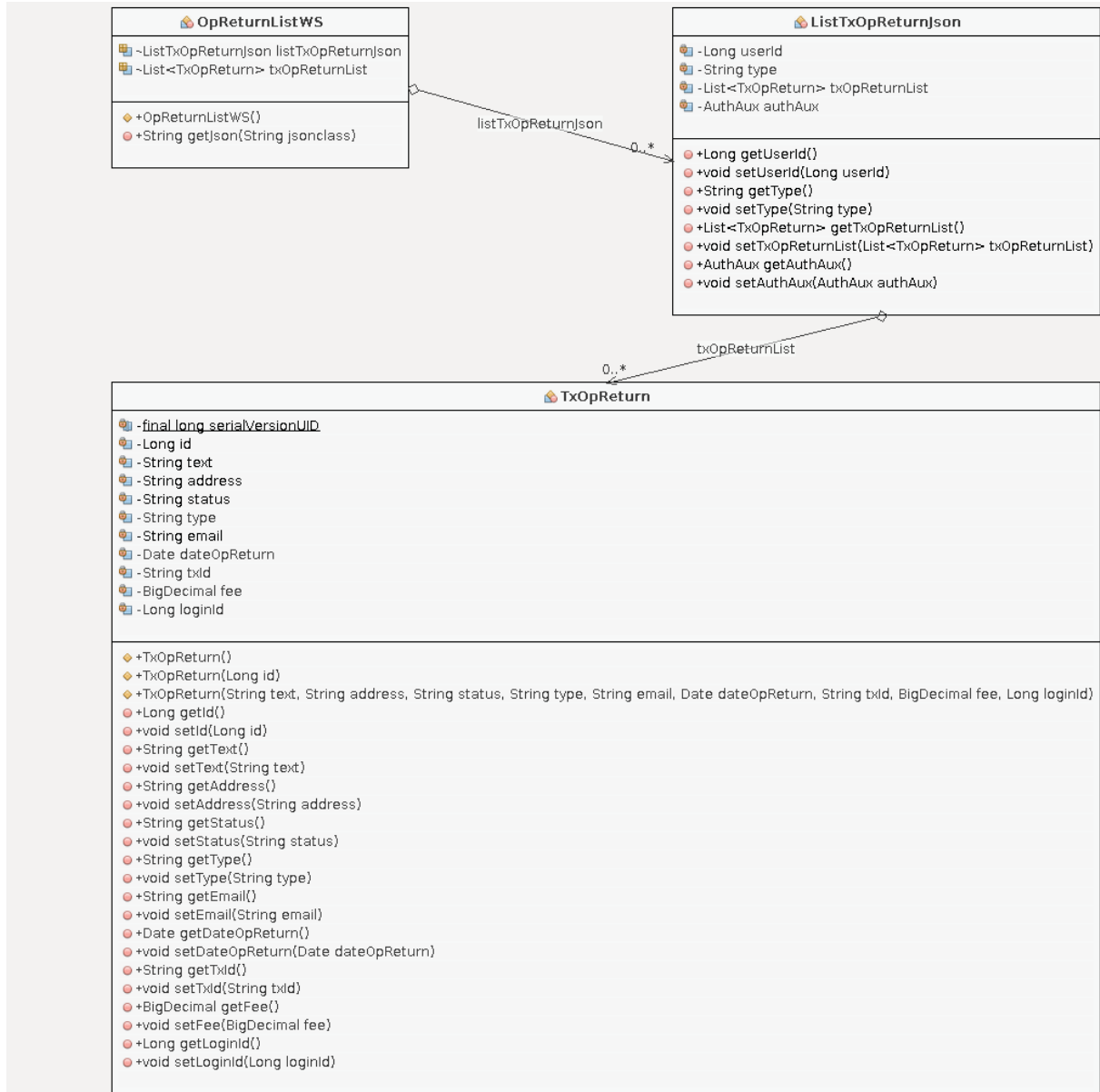
3.3.7.2.2 CLASS DIAGRAM OF VERIFICATION OF DOCUMENT AUTHENTICATION OPERATION ON SERVER APPLICATION

The class diagram in FIGURE 40 shows association of classes that are used in the process of verifying authentication of a signed document timestamped in the

⁶ A full size image of FIGURE 39 can be accessed in: https://github.com/apekato/serv/blob/master/src/main/resources/diagrams/CLASS_DIAGRAM_ANDROID_APP_VERIFICATION_OF_DOCUMENT_AUTHENTICATION.png

blockchain. This class diagram reads data from a database to get a transaction ID that is used to retrieve a signed document timestamped in the blockchain.

FIGURE 40 –VERIFICATION OF DOCUMENT AUTHENTICATION CLASS DIAGRAM



SOURCE: AUTHOR (2019)⁷.

Classes in the diagram of FIGURE 40 represent a static state of authentication verification of a signed document timestamped in the blockchain.

⁷ A full size image of FIGURE 40 can be accessed in: https://github.com/apekato/serv/blob/master/src/main/resources/diagrams/CLASS_DIAGRAM_VERIFICATION_OF_DOCUMENT_AUTHENTICATION.png

Class “OpReturnListWS” is associated directly and indirectly with all classes in the above diagram. This class makes method calls to read data from the database.

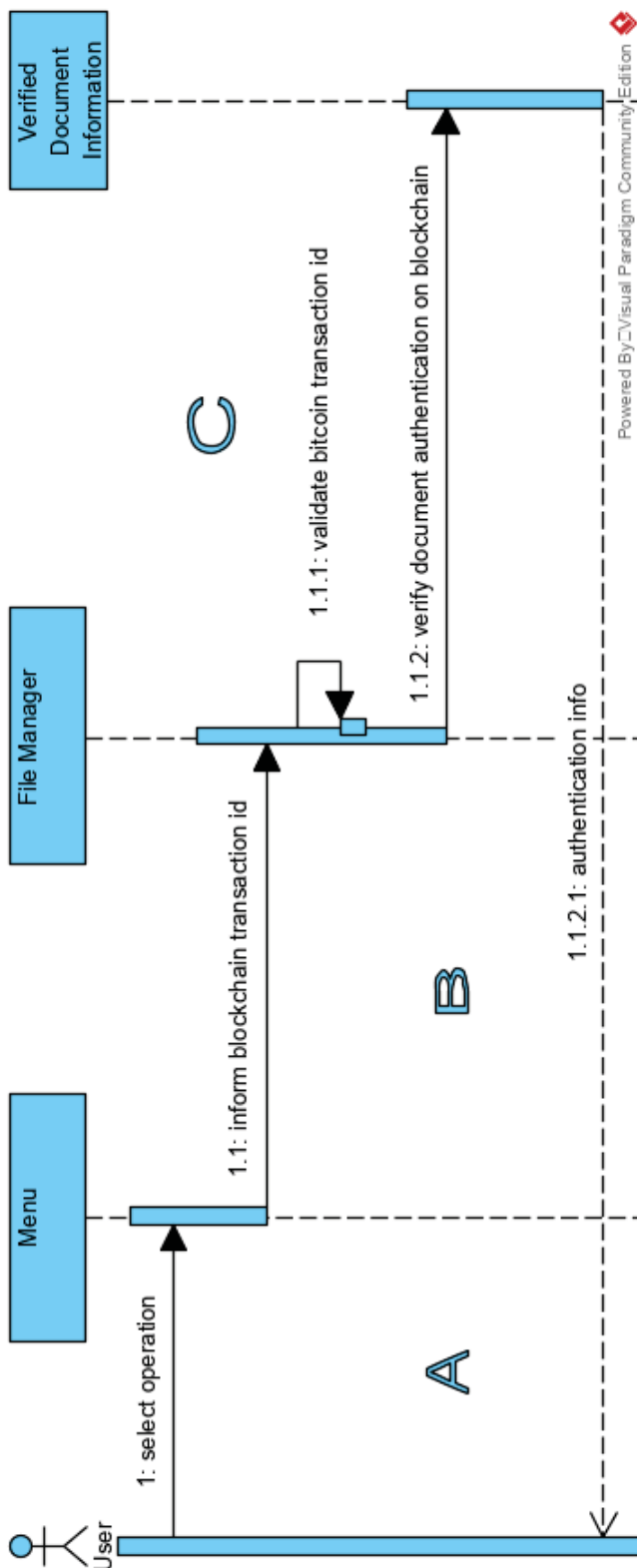
Class “OpReturnListWS” has an aggregation association with class “ListTxOpReturnJson”, i.e., class “ListTxOpReturnJson” is part of class “OpReturnListWS”. There is connectivity “one-to-many” from class “OpReturnListWS” to class “ListTxOpReturnJson”, i.e., class “ListTxOpReturnJson” returns a list of data of a user’s signed documents timestamped in the blockchain.

Class “ListTxOpReturnJson” has an aggregation association with class “TxOpReturn”, i.e., class “TxOpReturn” is part of class “ListTxOpReturnJson”. Class “TxOpReturn” is an exact model of database entity “tx_opreturn”, as it has the same attributes/variables as database entity “tx_opreturn”. There is connectivity “one-to-many” from class “ListTxOpReturnJson” to class “TxOpReturn”, i.e., class “TxOpReturn” returns a list of data of a user’s signed documents timestamped in the blockchain in form that are then translated into a JSON (JavaScript Object Notation) list of objects in class “ListTxOpReturnJson”.

3.3.7.3 SEQUENCE DIAGRAM OF VERIFICATION OF DOCUMENT AUTHENTICATION

The “Verification of Document Authentication” operation of the application prototype is described through a sequence diagram. FIGURE 41 presents a sequence diagram that describes the operations to verify authentication of a document signed with an ICP-Brasil digital certificate that was timestamped in the blockchain. The interactions between the objects are described using screenshots of the prototype's Android app.

FIGURE 41 – DOCUMENT AUTHENTICATION VERIFICATION SEQUENCE DIAGRAM



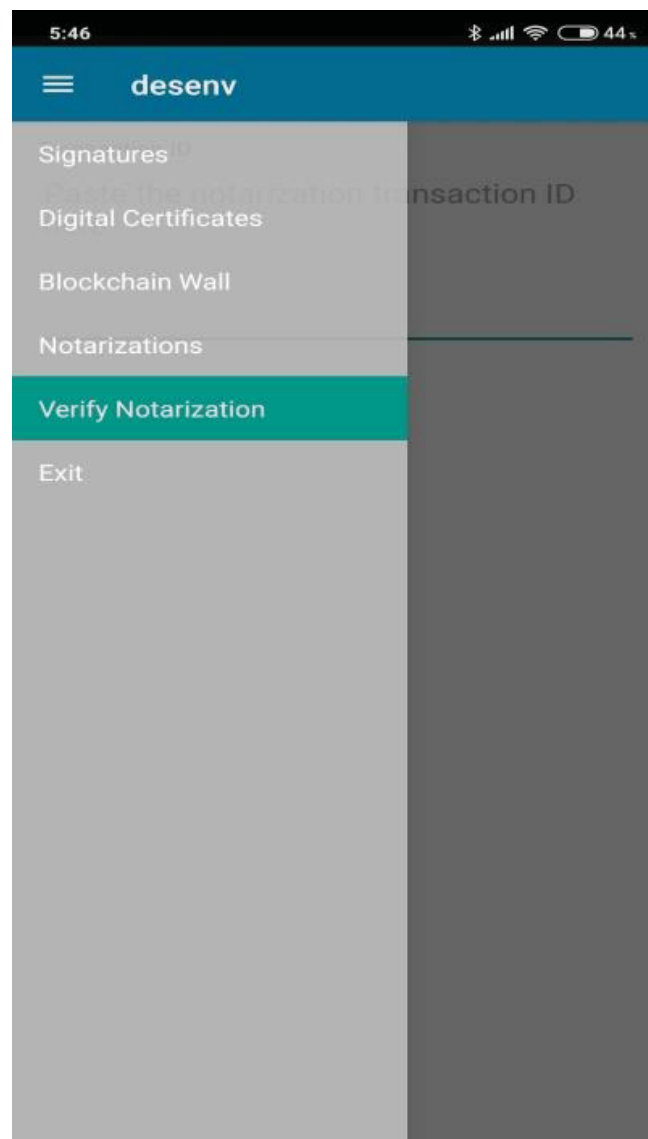
Powered By Visual Paradigm Community Edition



SOURCE: AUTHOR (2019).

FIGURE 42 illustrates part A (select operation) of the “Verification of Document Authentication” sequence diagram (FIGURE 41). This figure presents a menu of operations to the user, such as “Signatures”, “Digital Certificates”, “Notarizations”, and “Verify Notarization”. To initiate the operation of “Verification of Document Authentication”, the user taps on “Verify Notarization” on the menu of operations.

FIGURE 42 – FUNCTIONALITIES MENU ON ANDROID APP

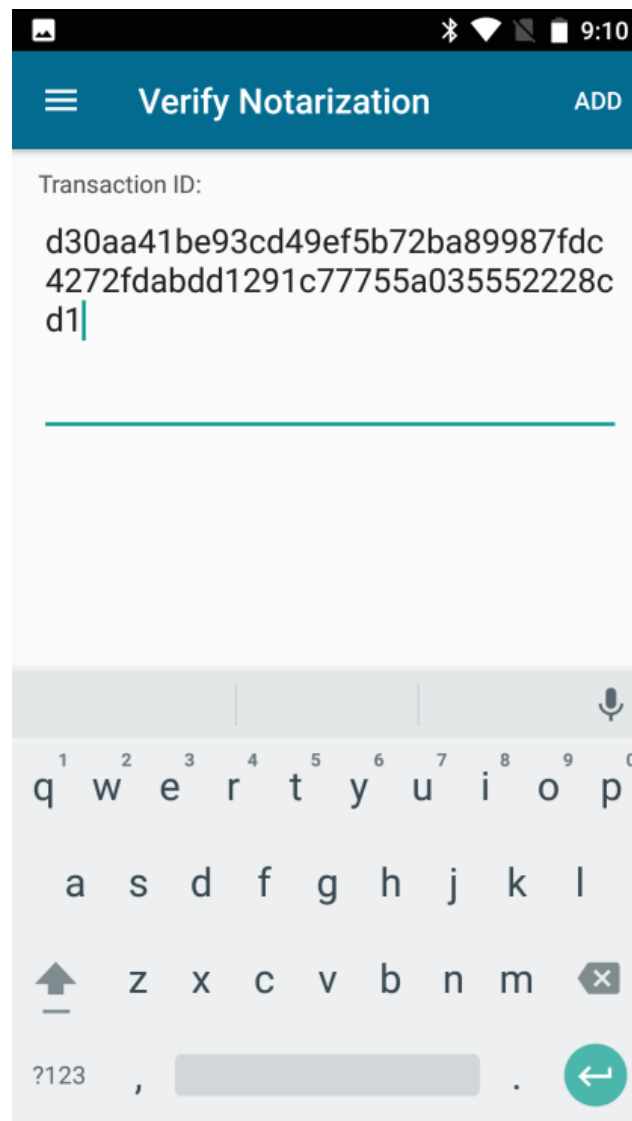


SOURCE: AUTHOR (2019).

FIGURE 43 illustrates part B (inform blockchain transaction id) of the “Verification of Document Authentication” sequence diagram (FIGURE 41). This figure presents the field “Transaction ID” the user needs to fill out to retrieve the document that was authenticated in the blockchain.

By tapping on “ADD” button, the app retrieves the transaction ID in the blockchain. If the transaction ID is successfully retrieved, it opens a File Manager to select a document the user wants to verify authentication. If it isn’t retrieved, the user is requested to reenter a valid transaction ID.

FIGURE 43 – TRANSACTION ID VERIFICATION

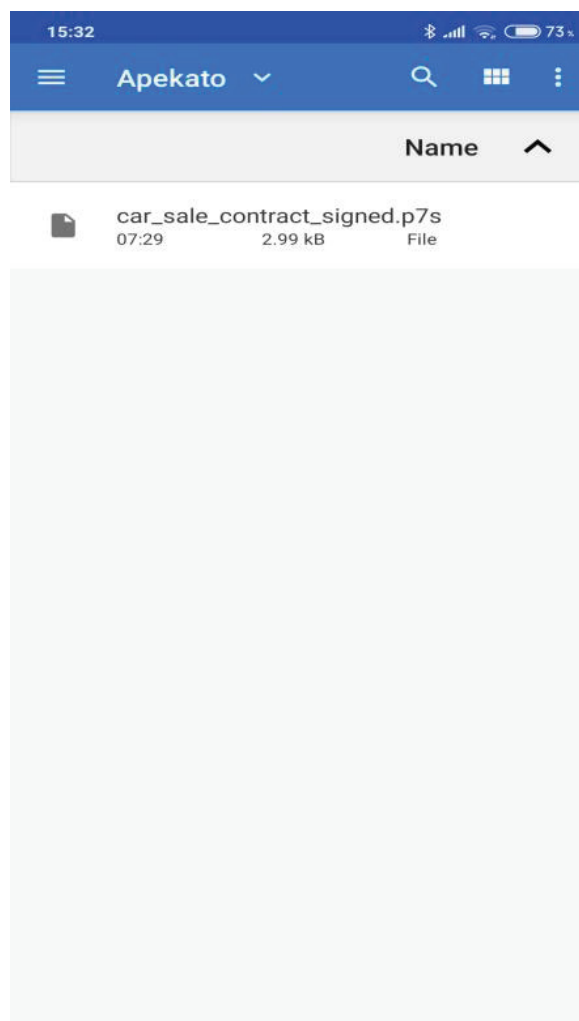


SOURCE: AUTHOR (2019).

FIGURE 44 illustrates part C (verify document authentication on blockchain) of the “Verification of Document Authentication” sequence diagram (FIGURE 41). This figure presents a File Manager to select a document signed with an ICP-Brasil digital certificate the user wants to verify authentication of a document signed with an ICP-Brasil digital certificate that was timestamped in the blockchain. The thumbnails list shows a list of documents signed with an ICP-Brasil digital certificate the user can select for verification of document authentication.

By tapping on a document signed with an ICP-Brasil digital certificate, the document is selected and the app automatically generates a SHA 256 Hash of the selected document. This SHA 256 Hash is compared to the SHA 256 Hash that was timestamped on the retrieved transaction ID which was informed on part B above.

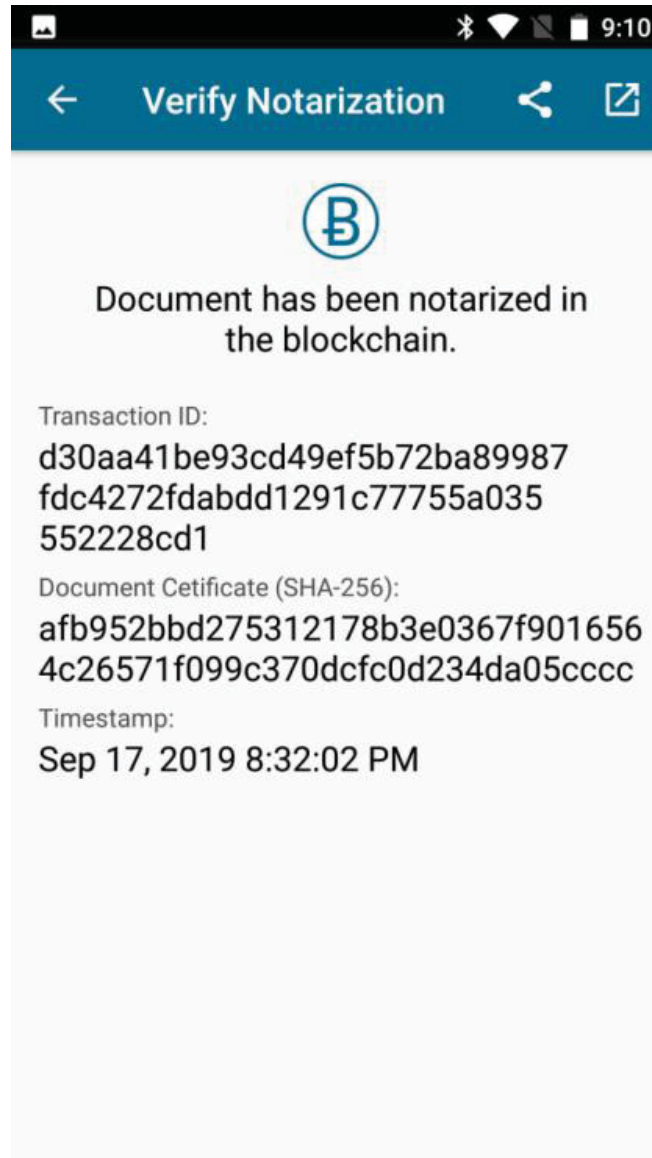
FIGURE 44 – FILE MANAGER TO SELECT SIGNED DOCUMENT FOR VERIFICATION OF AUTHENTICATION



SOURCE: AUTHOR (2019).

Finally, if the compared SHA 256 Hashes match each other, user gets information of the authenticated signed document (FIGURE 45). If the compared SHA 256 Hashes don't match each other, user gets information that the document was not found in the blockchain.

FIGURE 45 – VERIFIED DOCUMENT INFORMATION



SOURCE: AUTHOR (2019).

3.4 DEVELOPMENT OF APPLICATION PROTOTYPE

The application prototype was developed in the Java Programming Language. It is composed of an Android App, a Java Web Application Server. To develop the application prototype, the following open source codes were utilized: application server for Java Enterprise Edition platform called Glassfish (GLASSFISH, 2020), Java implementation of the Bitcoin protocol called bitcoinJ (BITCOINJ, 2017), Java implementation of digital signatures in the ICP-Brasil Standards called Demoseille Signer (DEMOISELLE SIGNER, 2019), and Android app implementation of blockchain timestamping utilizing Eternity Wall's Android client app (ETERNITY WALL, 2016).

The Java Web Application Server was built on a Glassfish Application Server, which was deployed on a Digital Ocean's Linux-based Virtual Machine (VM) (DIGITALOCEAN. Droplets, 2019). Digital Ocean "is a cloud computing vendor that offers an Infrastructure as a Service (IaaS) platform for software developers" (SEARCHCLOUDCOMPUTING. DigitalOcean, 2019).

BitcoinJ's implementation of the Bitcoin protocol can "maintain a wallet, send/receive transactions without needing a local copy of Bitcoin Core" (BITCOINJ, 2017), and it was deployed on a DigitalOcean's Linux-based VM.

Demoseille Signer provides functionalities for generating and validating ICP-Brasil digital signatures (DEMOISELLE SIGNER, 2019). The Demoseille Signer library is integrated into the Android app.

Eternity Wall's Android client app (ETERNITY WALL, 2016) provides services to register messages and timestamp files in the blockchain. Two of its implementation features were utilized in the Android App prototype: 1. Select and open internal storage files; 2. and send the user to other bitcoin wallet apps in order to pay for the service of timestamping files in the blockchain.

The Android app is the Graphical User Interface (GUI) for user interaction with the app's operations. Codification of each these operations are described next sections: import digital certificate (SECTION 3.4.1), document signing (SECTION 3.4.2), document authentication (SECTION 3.4.3), and verification of document authentication (SECTION 3.4.4).


```

622         File certFile = UtilsService.openFileInternalStorage(activity, StaticVars.DIGCERTLIST);
623         // load a digital certificates objects that may be stored in internal storage to a
624         // digital certificate object list
625         List<DigCert> digCertList = (List<DigCert>) UtilsService.getObjectFromFile(certFile);
626
627         // retrieve file name from uri path
628         String certName = UtilsService.retrieveFileName(activity, uri);
629
630         // open InputStream to load digital certificate
631         InputStream digCertLoadKeystoreStream =
activity.getContentResolver().openInputStream(uri);
632
633         // get digital certificate keystore if password is correct; otherwise, throw exception
634         KeyStore keyStore = KeyStorePKCS12.loadKeystore(activity, digCertLoadKeystoreStream,
pwd);
635         //
636         // open InputStream to convert it to bytes
637         InputStream digCertStream = activity.getContentResolver().openInputStream(uri);
638         byte[] digCertBytes = UtilsService.convertInputStreamToBytes(digCertStream);
639
640         // for verification of existing certificate in internal storage
641         boolean existDigCert = false;
642
643         // initialize digital certificate object list if empty
644         if (digCertList == null) {
645             digCertList = new ArrayList<>();
646             // if digital certificate object list is not empty, check digCertBytes sha256 hash matches
647             // an existing digCert
648         } else {
649
650             // load digital certificate from uri path into inputStream
651             InputStream digCertStreamSha256 = activity.getContentResolver().openInputStream(uri);
652             // get SHA 256 hash from digital certificate inputStream that was converted into bytes
653             // in method UtilsService.sha256Doc(activity, digCertStreamSha256)
654             String digCertSha256 = UtilsService.sha256Doc(activity, digCertStreamSha256);
655
656             // iterator to obtain a SHA 256 hash from each digital certificate in the list to compare
657             // with a SHA 256 hash from the digital certificate loaded in uri path
658             Iterator iterator = digCertList.iterator();
659             int index = 0;
660             while (iterator.hasNext()) {
661                 DigCert digCertIt = (DigCert) iterator.next();
662                 InputStream digCertItStream =
UtilsService.convertBytesToInputStream(digCertIt.getDigCertBytes());
663                 String digCertSha256Iterator = UtilsService.sha256Doc(activity, digCertItStream);
664                 if (digCertSha256.equals(digCertSha256Iterator)) {
665                     existDigCert = true;
666                 }
667             }
668         }
669         // save digital certificate from uri path in internal storage if verified that
670         // was not yet stored in internal storage
671         if (!existDigCert) {
672
673             DigCert digCert = KeyStorePKCS12.certInfo(KeyStorePKCS12.certicateChain(keyStore));
674             digCert.setFileName(certName);
675             digCert.setDigCertBytes(digCertBytes);
676             digCertList.add(digCert);
677             UtilsService.saveObjectInternalStorage(activity, digCertList, StaticVars.DIGCERTLIST);
678         }
679         inittItemFragment(StaticVars.TITLE_SECTION_DIGITAL_CERTIFICATE - 1);
680         uri = null;
681     } catch (FileNotFoundException e) {
682         e.printStackTrace();
683         uri = null;
684     } catch (IOException e) {
685         e.printStackTrace();
686         uri = null;
687     } catch (Exception e) {
688         e.printStackTrace();
689         uri = null;
690     }
691 }
692 })
693 .setNegativeButton(getText(R.string.action_cancel).toString(),
694     new DialogInterface.OnClickListener() {

```

```

695         public void onClick(DialogInterface dialog,
696                             int whichButton) {
697             dialog.dismiss();
698         }
699     }.show();
700
701     } else {
702         Toast toast = Toast.makeText(activity, activity.getString(R.string.toast_file_invalid), Toast.LENGTH_LONG);
703         toast.setGravity(Gravity.BOTTOM, 0, 0);
704         toast.show();
705     }
706
707 }
708 }

```

SOURCE: AUTHOR (2019).

3.4.2 DOCUMENT SIGNING CODIFICATION

Signing a document using a digital certificate is executed in the Android app through class “SignerAsyncTask” (APPENDIX 2). Execution of “Document Signing” operation is shown in CODE EXCERPT 2 from class “SignerAsyncTask”. Lines 61-71, loads the digital certificate into an inputStream variable from the Android app’s internal storage. In line 74, digital certificate is authenticated with the owner’s password and it becomes a KeyStore instance. The certificate private key is loaded from the KeyStore instance (line 77). A PKCS7Signer instance is created from the private key, which is utilized to sign a document (80). The document is loaded into a bytes array (lines 83-86) and it is signed in line 89. Finally, it is saved in the Android app’s public storage (lines 92-101).

CODE EXCERPT 2 – DOCUMENT SIGNING OPERATION

```

57     protected Boolean doInBackground(Void... voids) {
58
59         try {
60             // get list of digital certificates from internal storage
61             List<DigCert> digCertList = UtilsService.getDigCertList(mainActivity);
62
63             // get digital certificate in first position in the list, because as of now
64             // the app is coded to deal with only one digital certificate when signing documents
65             DigCert digCert = digCertList.get(0);
66
67             // get bytes format of digital certificate
68             byte[] digCertBytes = digCert.getDigCertBytes();
69
70             // convert bytes format of digital certificate into inputStream
71             InputStream digCertInputStream = UtilsService.convertBytesToInputStream(digCertBytes);
72
73             // get digital certificate keystore if password is correct; otherwise, throw exception
74             KeyStore keyStore = KeyStorePKCS12.loadKeystore(mainActivity, digCertInputStream, pwd);
75
76             // load private key from digital certificate
77             PrivateKey privateKey = KeyStorePKCS12.loadPrivKey(keyStore);
78
79             // get PKCS7 format for implementation of digital signatures
80             PKCS7Signer signer = KeyStorePKCS12.signer(keyStore, privateKey);
81

```

```

82 // load document from uri path into InputStream
83 InputStream inputStream = mainActivity.getContentResolver().openInputStream(uri);
84
85 // convert InputStream of document into bytes
86 byte[] fileBytes = UtilsService.convertInputStreamToBytes(inputStream);
87
88 // sign document and get bytes format of signed document
89 byte[] fileSignedBytes = signer.doDetachedSign(fileBytes);
90
91 // get name of signed document
92 String fileName = UtilsService.retrieveFileName(mainActivity, uri);
93
94 // create directory before attempting to a save file in a new directory
95 File directory = new File(Environment.getExternalStorageDirectory() + File.separator +
mainActivity.getString(R.string.app_name));
96 directory.mkdirs();
97
98 // save signed file to directory
99 String fileDirectorySignedPath = Environment.getExternalStorageDirectory() +
File.separator + mainActivity.getString(R.string.app_name) + File.separator + fileName +
"signedDetached_" + ".p7s";
100 UtilsService.savefileExternalStorage(fileSignedBytes, fileDirectorySignedPath);
101
102
103 } catch (ConnectException exception) {
104
105     glassfishDown = true;
106     return false;
107
108 } catch (CursorIndexOutOfBoundsException e) {
109     e.getMessage();
110 }
111 catch (Exception e) {
112     e.getMessage();
113     glassfishDown = true;
114     return false;
115 }

```

SOURCE: AUTHOR (2019).

3.4.3 DOCUMENT AUTHENTICATION CODIFICATION

Authentication of a signed document is executed in the main() method of class “OpReturnMain” (APPENDIX 3). This class utilizes an open source Java implementation of the Bitcoin protocol called bitcoinJ to process blockchain timestamp transactions through a script operation code called OP_RETURN.

OP_RETURN script invalidates a transaction output making it unspendable, but “allows a small amount of data to be inserted, which in our case is the document's [SHA 256] hash” (PROOF OF EXISTENCE, 2019).

CODE EXCERPT 3 of class “OpReturnMain” sets up the bitcoin wallet’s configurations. The implementation of the bitcoin wallet in the main() method of class “OpReturnMain” can use two distinct bitcoin blockchains: MainNet and TestNet. MainNet is used for real bitcoin transactions and TestNet is used for testing purposes by developers. For this project’s application prototype, only TestNet is used (line 55).

CODE EXCERPT 3 – MAINNET AND TESTNET BLOCKCHAINS

```

65 public static void main(String[] args) {
66
67     //
68     setupWalletKit();
69
70     // start a runnable thread process that runs the wallet's event listeners.
71     // It is constantly listening for events that occur to bitcoin addresses that belong to the wallet
72     bitcoin.addListener(new Service.Listener() {
73         @Override
74         public void starting() {
75             super.starting();
76             System.out.println("starting");
77         }
78
79         @Override
80         public void running() {
81             super.running();
82             System.out.println("running: " + bitcoin.wallet().currentChangeAddress().toString());
83         }
84
85         @Override
86         public void stopping(Service.State from) {
87             super.stopping(from);
88             System.out.println("stopping");
89         }
90
91         @Override
92         public void terminated(Service.State from) {
93             super.terminated(from);
94             System.out.println("terminated");
95         }
96
97         @Override
98         public void failed(Service.State from, Throwable failure) {
99             super.failed(from, failure);
100            System.out.println("failed");
101        }
102    }, Runnable::run);
103    bitcoin.addListener(new Service.Listener() {
104    }, OpReturnRunnable::runLater);
105    bitcoin.startAsync();
106
107
108 }
109
110 public static void setupWalletKit() {
111
112     // create new SPV (Simplified Payment Verification) bitcoinj app or
113     // if seed wallet is non-null it means we are restoring from backup.
114     bitcoin = new WalletAppKit(params, new File("."), TWININGS) {
115         @Override
116         protected void onSetupCompleted() {
117             // Don't make the user wait for confirmations for now, as the intention is they're sending it
118             // their own money!
119             bitcoin.wallet().allowSpendingUnconfirmedTransactions();
120             System.out.println("WalletAppKit onSetupCompleted: " + bitcoin.wallet().currentChangeAddress().toString());
121             System.out.println("port: " + params.getPort());
122             System.out.println("wallet current balance: " + bitcoin.wallet().getBalance().toString());
123
124             // Java Persistence API instance for application database
125             EntityManagerFactory emf = Persistence.createEntityManagerFactory("apekato");
126             EntityManager em = emf.createEntityManager();
127
128             try {
129                 // register invalid data when wallet is initialized and when wallet is changed
130                 registerInvalidData(em);
131             } catch (Exception ex) {
132                 Logger.getLogger(OpReturnMain.class.getName()).log(Level.SEVERE, null, ex);
133             }
134             em.close(); emf.close();
135
136             // start wallet event listeners
137             walletListener();
138         }

```

```

139     };
140 //     bitcoin.setBlockingStartup(false);
141 }

```

SOURCE: AUTHOR (2019).

The WalletAppKit instance in line 114 manages the bitcoin wallet. Through this wallet, bitcoin addresses are created, transactions are sent/received, and document timestamp transactions via OP_RETURN script are executed. Lines 72-106 start a runnable thread process that runs the wallet's event listeners. It is constantly listening for events that occur to bitcoin addresses that belong to the wallet.

CODE EXCERPT 4 from class "OpReturnMain" shows an event listener (lines 368 - 388) that is triggered when a wallet address receives bitcoins from a user who requests a signed document authentication service through the Android app by paying a transaction fee of ₦0.00015.

CODE EXCERPT 4 – PAYMENTS LISTENER

```

368     bitcoin.wallet().addCoinsReceivedEventListener(new WalletCoinsReceivedEventListener() {
369         @Override
370         public void onCoinsReceived(Wallet wallet, Transaction tx, Coin prevBalance, Coin newBalance) {
371             System.out.println("onCoinsReceived");
372
373             EntityManagerFactory emf = Persistence.createEntityManagerFactory("apekato");
374             EntityManager em = emf.createEntityManager();
375
376             try {
377                 registerOpReturnData(em, tx);
378                 // register invalid data when wallet is initialized and when wallet is changed
379                 registerInvalidData(em);
380             } catch (Exception ex) {
381                 Logger.getLogger(OpReturnMain.class.getName()).log(Level.SEVERE, null, ex);
382             }
383
384             em.close();
385             emf.close();
386         });
387     }
388 }

```

SOURCE: AUTHOR (2019).

Line 377 calls a method that retrieves a wallet address that has received a bitcoin payment which was registered on the application database. Next, it calls another method, shown in CODE EXCERPT 5, that timestamps a signed document's SHA 256 Hash through the OP_RETURN script operation code (lines 240 -252), and updates the status of a signed document's timestamp process to "REGISTERED" on the application database (277-281).

CODE EXCERPT 5 – TIMESTAMP DATA

```

225 public static void timestampData(EntityManager em, OpReturn opReturn) throws IOException,
InsufficientMoneyException, Exception {
226
227 // create a byte variable to convert a SHA 256 hash of a signed document into bytes
228 byte[] opReturnBytes = null;
229
230 // timestamp any text message in the blockchain
231 if (opReturn.getType().endsWith(OpReturn.OpReturnType.OP_RETURN_TYPE_TEXT)){
232     opReturnBytes = opReturn.getText().getBytes("UTF-8");
233
234 // timestamp an ICP-Brasil signed document in the blockchain
235 } else if (opReturn.getType().endsWith(OpReturn.OpReturnType.OP_RETURN_TYPE_NOTARIZATION)){
236     opReturnBytes = Hex.decode(opReturn.getText());
237 }
238
239 // Create a tx with an OP_RETURN output
240 Transaction tx = new Transaction(params);
241 tx.addOutput(Coin.ZERO, ScriptBuilder.createOpReturnScript(opReturnBytes));
242
243 System.out.println("wallet before tx: " + bitcoin.wallet().getBalance().toString());
244
245 // send wallet information regarding timestamping data.
246 SendRequest req = SendRequest.forTx(tx);
247
248 // set timestamp transaction fee (0.00015 BTC)
249 BigDecimal sendfee = opReturn.getFee().setScale(5, RoundingMode.HALF_EVEN);
250
251 // send timestamp transaction
252 req.feePerKb = Coin.parseCoin(sendfee.toString());
253
254 // Coin c = req.feePerKb;
255 // if (c.value < 15000) {
256 //     long add = 15000 - c.value;
257 //     req.feePerKb.add(Coin.valueOf(add));
258 // } else if (c.value > 15000) {
259 //     long subtract = c.value - 15000;
260 //     req.feePerKb.add(Coin.valueOf(subtract));
261 // }
262 // Coin c2 = req.feePerKb;
263 // Send it to the Bitcoin network
264
265 // get result information of timestamp transaction
266 Wallet.SendResult sendResult = bitcoin.wallet().sendCoins(req);
267
268 // long fee = sendResult.tx.getFee().longValue();
269
270 System.out.println("getHashAsString: " + sendResult.tx.getHashAsString());
271
272 // checks if timestamp transaction was successful
273 if (sendResult.tx.getHashAsString() != null){
274
275     // set timestamp opReturn object for persistence in application database
276     // change timestamp opReturn object status to REGISTERED
277     opReturn.setStatus(OpReturn.OpReturnStatus.OP_RETURN_STATUS_REGISTERED);
278     // set timestamp transaction ID into timestamp opReturn object
279     opReturn.setTxId(sendResult.tx.getHashAsString());
280     // persist timestamp opReturn object in application database
281     String ok = GenericDaoJpa.updateWithoutTx(em, OpReturn.class, opReturn);

```

SOURCE: AUTHOR (2019).

3.4.4 VERIFICATION OF DOCUMENT AUTHENTICATION CODIFICATION

Verifying authentication of a signed document is executed in the Android app through class “VerifyNotarizationAsyncTask” (APPENDIX 4). Execution of “Verification of Document Authentication” operation is shown in CODE EXCERPT 6

from class “VerifyNotarizationAsyncTask”. Line 84 makes an http request utilizing BlockCypher’s API (BLOCKCYPHER, 2019) to retrieve information of a timestamp transaction on bitcoin’s blockchain. On the Android app, the user informs the “Transaction ID” of the timestamped document, which is passed as parameter along with the http request.

If the timestamp transaction is successfully retrieved, the SHA 256 Hash that was timestamped in the blockchain is compared to the SHA 256 Hash calculated from the signed document. Finally, if the compared SHA 256 Hashes match each other, user receives information of the authenticated signed document.

CODE EXCERPT 6 – VERIFICATION OF DOCUMENT AUTHENTICATION

```

78  protected Boolean doInBackground(String... params) {
79
80      try {
81
82          // url to retrieve a transaction id of a document timestamped
83          // on bitcoin's blockchain using BlockCypher's API
84          String url_ = ProjService.BLOCK_CYPHER_ENDPOINT + "txs/" + txId + "?token=" +
ProjService.BLOCK_CYPHER_TOKEN;
85
86          // get a pointer to a "resource" on the World Wide Web
87          URL url = new URL(url_);
88
89          // send request to BlockCypher's API
90          HttpURLConnection urlConnection = (HttpURLConnection)url.openConnection();
91
92          // load into InputStream data of timestamp transaction on bitcoin's blockchain
93          InputStream in = urlConnection.getInputStream();
94
95          // convert InputStream to InputStreamReader
96          InputStreamReader reader =
97              new InputStreamReader(in);
98
99          // load Tx (transaction) object that has detailed information of transaction id of
100         // a document timestamped on bitcoin's blockchain using BlockCypher's API
101         tx = new Gson().fromJson(reader, Tx.class);
102
103     } catch (FileNotFoundException exception) {
104         fileNotFoundException = true;
105         return false;
106     } catch (IOException exception) {
107         iOExceptionWrongFormat = true;
108         return false;
109     } catch (Exception exception) {
110         exception.printStackTrace(); // show exception details
111         return false;
112     }
113
114     return true;
115 }

```

SOURCE: AUTHOR (2019).

4 RESULTS AND DISCUSSIONS

In comparison with related works presented in SECTION 2.9, this section analyses implementations of each related work (TABLE 3) in regards to the following features: identity association, Certificate Revocation List (CRL) and blockchain timestamp. This section presents advantages, disadvantages and issues related to the application prototype's use of bitcoin blockchain and ICP-Brasil digital certificates in comparison with the use of other technologies used by related works.

4.1 IDENTITY ASSOCIATION

Identity association among related works shown in TABLE 3 has two implementation methods: digital certificate and blockchain ID. The digital certificate method was implemented by this research project's application prototype and the company VALIDChain. The blockchain ID method was implemented by the company OriginalMy. And the open standard Blockcerts allows implementation of either digital certificate or blockchain ID methods due to its flexible design.

4.1.1 ADVANTAGE OF DIGITAL CERTIFICATES

The advantage of digital certificates over blockchain IDs is related to security of Know Your Customer (KYC) process. The digital certificate method is more secure than the blockchain ID method because this application prototype and VALIDChain's digital certificates are implementations of the Brazilian Public Key Infrastructure (ICP-Brasil), which requires in-person identification and physical proofs of identity and address documents to ICP-Brasil's Registration Authorities (RA) officials. On the other hand, OriginalMY's blockchain ID implementation requires online submission of proof of identity through its mobile app.

4.1.2 DISADVANTAGE OF DIGITAL CERTIFICATES

The disadvantage of ICP-Brasil's digital certificates over blockchain IDs is related to geographic delimitation restricted on the region of Brazil and costs of issuing digital certificates. ICP-Brasil's digital certificates are issued only to Brazilian

citizens and digital documents signed with ICP-Brasil's certificates are only valid in Brazil through regulatory Provisional Measure 2.200-2/2001 that created ICP-Brasil's digital certification. On the other hand, OriginalMY's blockchain IDs do not have jurisdictional geographic delimitation regarding associating identities to blockchain IDs. This allows issuance of blockchain IDs to citizens of any nationality; and it allows worldwide acceptance of digital documents signed with blockchain IDs as long as admitted by countries as valid/legal.

The costs of issuing ICP-Brasil digital certificates in file format, excluding token, card, mobile and cloud types of digital certificates, on a popular Certification Authority in Brazil called Certisign are presented in TABLE 4:

TABLE 4 – COST OF ISSUING DIGITAL CERTIFICATES ON CERTISIGN AS OF MAY 23, 2020

VALIDITY	NATURAL PERSON	LEGAL ENTITY
1 YEAR	R\$ 168	R\$252
2 YEARS	-	R\$ 302
3 YEARS	R\$ 252	R\$ 352

SOURCE: CERTISIGN (2020).

The high costs of issuing digital certificates may hinder its adoption and usage to digitally sign documents among the Brazilian population. As a result, digital certificates might end up being used mostly to access government online systems, such as eSocial and Empregador (Employer) Web for employers to report work related information on their employees, eCAC for companies and individuals to file income tax return, and digitally sign medical prescriptions during the COVID-19 pandemic.

4.2 REVOCATION LIST (RL)

Revocation List (RL) among related works shown in TABLE 3 has two implementation methods: Certificate Revocation List (CRL) and "blockchain ID unsubscription". ICP-Brasil's CRL is implemented by this research project's application prototype though the open source Java implementation of digital signatures in the ICP-Brasil Standards called Demoseille Signer. And it is assumed

that the company VALIDChain implements ICP-Brazil's CRL since it is a Certification Authority (CA) which provides digital signatures according to ICP-Brazil's standards.

The company OriginalMy provides a method similar to a CRL in order to revoke blockchain IDs. If a blockchain ID password is lost or stolen, it can be unsubscribed and a new blockchain ID can be obtained through OriginalMY's mobile app. The open standard Blockcerts allows flexible implementation to RLs.

4.2.1 ISSUE WITH DIGITAL IDENTITIES

Including a hacked or stolen digital identity in a Revocation List (RL) does not block it from signing documents since execution of digital signatures works independently from RLs.

In the ICP-Brazil system, a revoked digital certificate is still able to perform valid signatures on documents without the need to verify online inclusion of the digital certificate in ICP-Brazil's Certificate Revocation List (CRL), as long as the digital signature implementation follows ICP-Brazil Standards. Additionally, a revoked blockchain ID, included in a RL, is also able to perform valid signatures on documents (and send/receive coins) without the need to verify online inclusion of the blockchain ID in a RL.

A proper implementation of digital signatures can be ensured by verifying online if a digital identity was included in a RL database prior to signing a document. If it is confirmed that a digital identity has been revoked, a digital signature should not be performed on a document.

4.3 BLOCKCHAIN TIMESTAMP PERFORMANCE

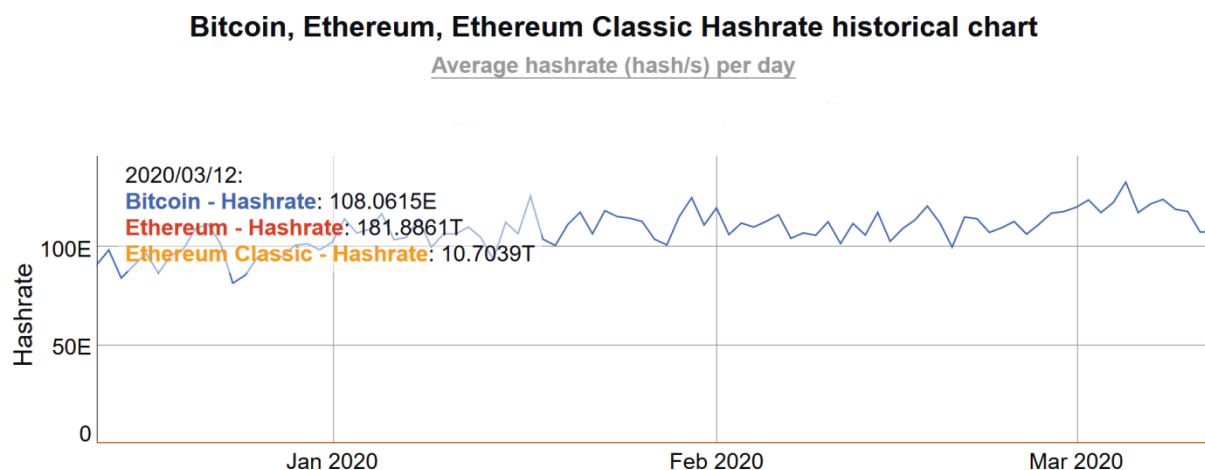
An analysis on blockchain timestamp performance compares blockchains utilized in related services shown in TABLE 3: bitcoin, ethereum and ethereum classic. Three factors are analyzed for performance comparison:

- average hashrate per day (FIGURE 47): measures network security;
- average block timestamp intervals (FIGURE 46): measures transaction timestamp speeds;

- average transaction fee per day (FIGURE 48 and FIGURE 49): measures cost of timestamp transactions.

4.3.1 AVERAGE HASHRATE PER DAY

FIGURE 46 – BITCOIN - ETHEREUM – ETHEREUM CLASSIC AVERAGE HASHRATE PER DAY



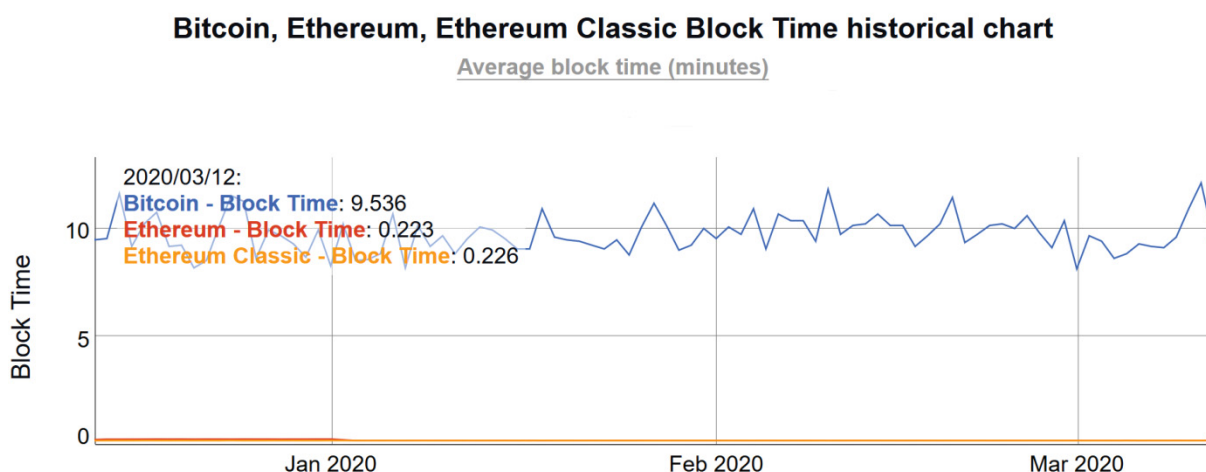
SOURCE: BITINFOCHARTS (2020a).

A advantage of bitcoin over ethereum and ethereum classic is related to its higher average hashrate that makes it more secure in preventing double-spending attacks (SECTION 2.1.3) on the bitcoin network through proof-of-work (SECTION 2.1.4), since the average hashrate of 108.0615E is 594 times higher than ethereum and 10,095,526 times higher than ethereum classic's average hashrate measured, all measured on March 12, 2020 FIGURE (46).

Bitcoin's average hashrate per day on March 12, 2020 was 108.0615E (108.0615 quintillion hashes or exahashes per second), i.e., 108,061,500,000,000,000,000 hashes per second. Ethereum's average hashrate per day on March 12, 2020 was 181.8861T (181.8861 trillion hashes or terahashes per second), i.e., 181,886,100,000,000,000 hashes per second. Ethereum Classic's average hashrate per day on March 12, 2020 was 10.7039T (10.7039 trillion hashes or terahashes per second), i.e., 10,703,900,000,000 hashes per second (FIGURE 46).

4.3.2 AVERAGE BLOCK TIMESTAMP INTERVALS

FIGURE 47 – BITCOIN - ETHEREUM – ETHEREUM CLASSIC AVERAGE BLOCK TIMESTAMP INTERVALS IN MINUTES



SOURCE: BITINFOCHARTS (2020b).

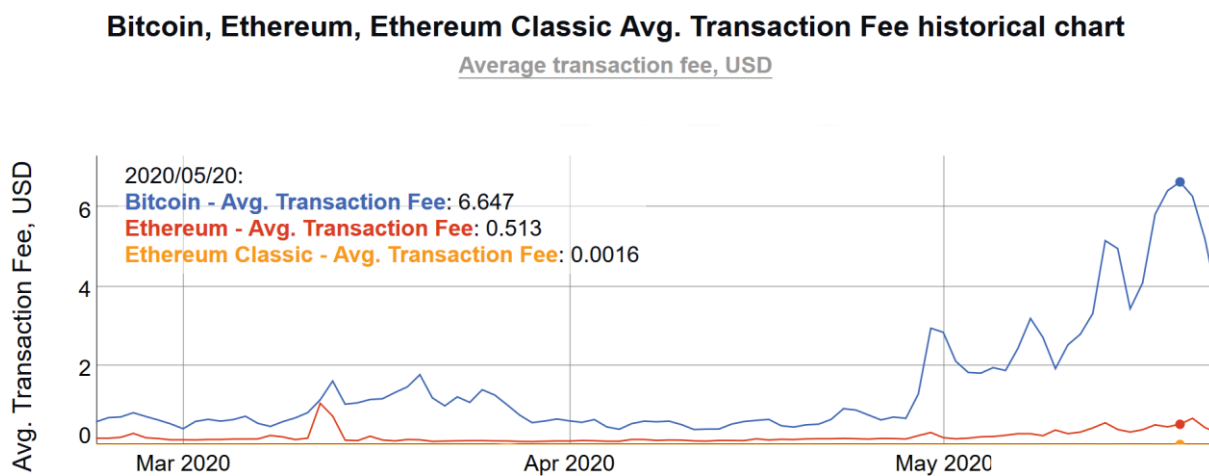
A disadvantage in utilizing bitcoin blockchain is related to its longer average block timestamp intervals. FIGURE 47 shows that in order to obtain a bitcoin timestamp it takes approximately 21.23 times longer than ethereum and ethereum classic, according to the average block timestamp interval in minutes measured on March 12, 2020.

Bitcoin's average block timestamp interval in minutes on March 12, 2020 was 9.536 minutes. Ethereum's average block timestamp interval in minutes on March 12, 2020 was 0.223 minutes. Ethereum classic's average block timestamp interval in minutes on March 12, 2020 was 0.226 minutes (FIGURE 47).

4.3.3 AVERAGE TRANSACTION FEE

Another disadvantage with bitcoin blockchain is related to its volatile average transaction fee, which greatly increases when the bitcoin price and volume of transactions are high. Transaction fees are paid to miners, who can prioritize transactions included in a block based on higher transaction fees. If the average transaction fee is high, it increases the time to obtain a bitcoin timestamp (authentication) on a document signed with an ICP-Brasil digital certificate using the application prototype since the transaction fee is set to ₺0.00015.

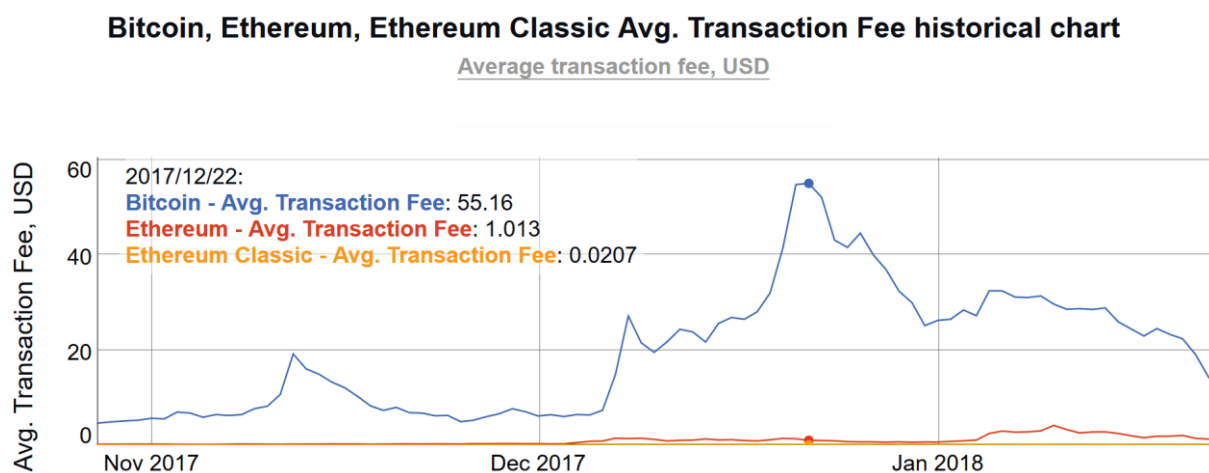
Figure 48 – BITCOIN – ETHEREUM - ETHEREUM CLASSIC AVERAGE TRANSACTION FEE PER DAY A



SOURCE: BITINFOCHARTS (2020c).

On May 20, 2020 (FIGURE 48), the average transaction fee of bitcoin was US\$ 6.64, of ethereum was US\$ 0.513, and of ethereum classic was US\$ 0.0016, whereas, in comparison, the average transaction fee on December 22, 2017 (FIGURE 49) was approximately eight times higher on bitcoin at US\$ 55.16, approximately twice as higher on ethereum at US\$ 1.013, and approximately twelve times higher on ethereum classic at US\$ 0.0207.

Figure 49 – BITCOIN – ETHEREUM - ETHEREUM CLASSIC AVERAGE TRANSACTION FEE PER DAY B



SOURCE: BITINFOCHARTS (2020c).

On May 20, 2020, with the price of one bitcoin at US\$ 9,677.00 (BITINFOCHARTS, 2020d), the application prototype transaction fee of $\text{฿}0.00015$ in US dollars was US\$ 1.45, which was approximately 5 times lower than the bitcoin average transaction fee at US\$ 6.64.

On December 22, 2017, with the price of one bitcoin at US\$ 14,030.00 (BITINFOCHARTS, 2020d), the application prototype transaction fee of $\text{฿}0.00015$ in US dollars was US\$ 2.1, which was approximately 26 times lower than the bitcoin average transaction fee at US\$ 55.16.

Comparing average transaction fees between blockchains, the chart on FIGURE 48 shows that average transaction fee of bitcoin at US\$ 6.64 was approximately thirteen times higher than ethereum's at US\$ 0.513, approximately 4 thousand times higher than ethereum classic's at US\$ 0.0016.

5 CONCLUSIONS

This chapter presents research considerations related to its objectives, final considerations and recommendations for future work regarding the research study.

5.1 REGARDING RESEARCH OBJECTIVES

Regarding this research project's general objective (SECTION 1.3.1), it integrated the dichotomy between a centralized ICP-Brasil versus a decentralized blockchain. Integration of both technologies simplifies the development of a digital document authentication service since both are outsourced services.

ICP-Brasil provides an outsourced KYC (Know Your Customer) service by verifying through a Registration Authority the user's identity and address, prior to issuing a digital certificate to the user. And bitcoin's blockchain provides an outsourced tamper-proofing timestamp data since it is a peer-to-peer network built on the internet that surpasses geographic boundaries.

Regarding this research project's specific objectives (SECTION 1.3.2), three specific objectives were analyzed for this research project (SECTION 1.3.2): information security, application prototype system requirements, and application prototype development.

This research identified how integration of blockchain timestamping and digital signatures based on ICP-Brasil digital certificates provides a digital document authentication service that assures information security through analysis of the following concepts: authenticity, availability, confidentiality, identity, immutability, integrity, legality and non-repudiation.

It was observed that blockchain timestamping by itself is able to provide proof of existence of documents since it guarantees data integrity and immutability. Through the principle of data integrity, a hashed document timestamped in the blockchain is prevented from being changed. This is due to the principle of immutability, which prevents a hashed document timestamped in the blockchain from being corrupted due to the proof of work that requires unaffordable CPU processing resources to roll back the blockchain records.

ICP-Brasil provides additional principles of information security such as identity, legality, and non-repudiation. The principle of identity is provided through ICP-Brasil digital certificates that are equivalent to a digital identity. The principle of legality is provided through regulatory Provisional Measure 2.200-2/2001 which institutes that ICP-Brasil digital signatures have legal validity equivalent to a document signed on paper. And ICP-Brasil digital signatures provide the principle of non-repudiation since it prevents a user from denying having signed a document because identity association is processed by identifying a user to an ICP-Brasil Certification Authority, who, in turn, assures that the user owns the digital certificate.

Authenticity of a document is proven in terms of data integrity through blockchain timestamping, and in terms of identity through digital signatures based on ICP-Brasil digital certificates. In regards to the principle of availability, blockchain's distributed peer-to-peer system assures availability of its data structure that allows registration and verification of timestamping documents. ICP-Brasil assures availability of its Certificate Revocation List (CRL), which should always be verified online if an ICP-Brasil digital certificate has been revoked prior to signing a document. In regards to the principle of confidentiality, only a SHA 256 message digest of a document signed with an ICP-Brasil digital certificate is registered in the blockchain to obtain a timestamp.

This research identified system requirements (SECTION 3.3) to develop an application prototype for mobile Android devices that integrates of blockchain timestamping and digital signatures based on ICP-Brasil digital certificates.

Four operations were analyzed for proper operation of the prototype: import digital certificate, document signing, document authentication, and document authentication verification. For each of these operations, the Unified Modeling Language (UML) was utilized to specify case diagrams, sequence diagrams, class diagrams.

Additionally, UML entity-relationship diagrams were utilized in order to specify a database for the application prototype. This database registers user's data for login authentication into the application, registers bitcoin wallet addresses that receive payment fees from users who timestamp documents signed with an ICP-Brasil digital certificate through the app, and registers transaction IDs that are used to retrieve a SHA 256 message digest of a document signed with an ICP-Brasil digital certificate in the blockchain.

This research followed system requirements identified in the previous specific objective (SECTION 5.2.2) in order to develop an application prototype for mobile Android devices that integrates blockchain timestamping and digital signatures based on ICP-Brasil digital certificates. The following operations were built into application prototype: import ICP-Brasil digital certificates in the mobile Android device; digital signature of documents using digital certificates issued by an ICP-Brasil Certification Authority; blockchain timestamping of documents that were digitally signed with ICP-Brasil certificates; blockchain timestamp verification of documents for proof of authenticity.

The application prototype is comprised of an Android app and server application developed in the Java programming language. The source code of the application prototype was published on GitHub under the open source MIT License, and can be accessed on:

- Android app: <https://github.com/apekato/app>
- Server application: <https://github.com/apekato/serv>

5.2 FINAL CONSIDERATIONS

This research project develops a prototype for a digital document authentication service by developing an application prototype for mobile Android devices that integrates blockchain timestamping and digital signatures based on ICP-Brasil digital certificates. Recently, because of the COVID-19 pandemic that has spread in Brazil, doctors are now allowed by the Health Ministry to issue medical prescriptions signed with ICP-Brasil digital certificates (SECTION 1.2), which shows increased demand to digitalize the notarial system in Brazil. It also points out the relevance of developing apps for mobile devices that provide digital document authentication services.

The literature reviewed in this project gives an overview of how Bitcoin blockchain and ICP-Brasil certificates technologies function and how both can be integrated to provide a digital document authentication service. It was observed that information security is assured considering concepts of authenticity, availability, confidentiality, identity, immutability, integrity, legality and non-repudiation. In comparison with related works (SECTION 2.9) that provide services similar to this

project's research, it was analyzed in the results (SECTION 4) advantages, disadvantages and issues related to integration of digital certificates and blockchain timestamping.

In regards to advantages of utilizing both technologies, identity association with digital certificates presented a secure method of identity verification through KYC (Know Your Customer) which requires in-person identification and physical proofs of identity and address documents to ICP-Brasil's Registration Authorities (RA) officials. Bitcoin's higher hashrate than ethereum and ethereum classic makes its blockchain more secure in preventing double-spending attacks. On the other hand, disadvantages regarding integration of both technologies have been presented that are indicated as recommendations for future work in SECTION 5.3.

5.3 RECOMMENDATIONS FOR FUTURE WORK

Recommendations for future work are divided as:

- Future research topics regarding disadvantages that have been presented in integrating digital certificates and blockchain to provide a digital document authentication service;
- Tools to increment functionalities of the application prototype for mobile Android devices that integrates blockchain timestamping and digital signatures based on ICP-Brasil digital certificates.

In regards to disadvantages in integrating digital certificates and blockchain, the following recommendations for future research are presented:

- Due to geographic delimitation restricted on the region of Brazil and high cost of issuing ICP-Brasil digital certificates, extended research may be conducted in utilizing blockchain IDs as substitute to digital certificates in order to associate identities of citizens and legal entities of any nationalities. Similar work is already being done utilizing blockchain IDs by company OriginalMY;
- Due to bitcoin's longer average block timestamp intervals and higher transaction fees, extended research may be conducted in utilizing

blockchains that have shorter average block timestamp intervals and lower transaction fees, taking into consideration a blockchain's network security: high hashrate, proof-of-work consensus and decentralization, i.e., worldwide adoption of the network.

In regards to tools to increment functionalities, the following recommendations for future work are presented:

- Tool to import more than one digital certificate into the Android app. This would allow a user to import its natural person and legal entities ICP- Brasil digital certificates into the app;
- Tool to allow multiple signatures on a single document, so that a contract with two or more parties, represented by natural persons or legal entities, can sign a single document with ICP-Brasil digital certificates;
- Tool to allow blockchain timestamping of multiple digital signatures in a single transaction, utilizing merkle trees which recursively summarizes N elements (digital signatures) into a single hash to be timestamped in the blockchain with a single transaction;
- Tool applied to supply chain processes to record, transmit, and share data authentication securely.
- Tool to allow users to prioritize confirmation of document timestamps in the bitcoin blockchain by choosing between Priority fee, Normal fee and Economic fee, instead of the current fixed fee of ₺0.00015 the application prototype.

REFERENCES

ANDRESS, J. **The Basics of Information Security: Understanding the Fundamentals of Infosec in Theory and Practice.** Waltham, MA, USA: Elsevier, 2014.

_____. **The bitcoin and open blockchain expert: About.** Available in: <https://aantonop.com/bio/>. Accessed on: 22 May. 2020.

ANTONOPOULOS, A. **Mastering Bitcoin: Unlocking Digital Crypto-Currencies.** California, USA: O'Reilly Media, 2014.

ANTONOPOULOS, A.; WOOD, G. **Mastering Ethereum: Building Smart Contracts and DApps.** California, USA: O'Reilly Media, 2018.

ANVISA. **COVID-19 - Medicamentos controlados: receitas com assinatura digital.** Available in: http://portal.anvisa.gov.br/noticias/-/asset_publisher/FXrpx9qY7FbU/content/medicamentos-controlados-receitas-com-assinatura-digital/219201/pop_up?inheritRedirect=false&redirect=http%3A%2F%2Fportal.anvisa.gov.br%2Fnoticias%3Fp_p_id%3D101_INSTANCE_FXrpx9qY7FbU%26p_p_lifecycle%3D0%26p_p_state%3Dpop_up%26p_p_mode%3Dview%26p_r_p_564233524_tag%3Dcovid-19. Accessed on: 21 May. 2020.

BASHIR, I. **Mastering Blockchain: Distributed ledgers, decentralization and smart contracts explained.** Birmingham, United Kingdom: Packt Publishing, 2017.

BEAVER, K; **Hacking For Dummies.** Hoboken, New Jersey, USA: John Wiley & Sons, 2016.

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML.** São Paulo, SP, Brazil: Elsevier, 2015.

BITCOINJ. A library for working with Bitcoin. Version 0.14.4 [S.I.]: bitcoinj, Feb. 07, 2017. Available in: <<https://github.com/bitcoinj/bitcoinj/tree/v0.14.4>>. Accessed on: 07 Jul. 2020.

BITINFOCHARTS. Bitcoin, Ethereum, Ethereum Classic Hashrate historical chart. 2020a. Available in: <<https://bitinfocharts.com/comparison/hashrate-btc-eth-etc.html#3m>>. Accessed on: 13 Mar. 2020.

_____. **Bitcoin, Ethereum, Ethereum Classic Block Time historical chart.** 2020b. Available in: <<https://bitinfocharts.com/comparison/confirmationtime-btc-eth-etc.html#3m>>. Accessed on: 13 Mar. 2020.

_____. **Bitcoin, Ethereum, Ethereum Classic Avg. Transaction Fee historical chart.** 2020c. Available in: <<https://bitinfocharts.com/comparison/transactionfees-btc-eth-etc.html#3m>>. Accessed on: 25 May. 2020.

_____. **Bitcoin (BTC) price stats and information.** 2020d. Available in: <<https://bitinfocharts.com/bitcoin/>>. Accessed on: 07 Jul. 2020.

BLOCKCERTS. Introduction. 2020a. Available in: <<https://www.blockcerts.org/guide/>>. Accessed on: 14 Jan. 2020.

_____. **FAQ.** 2020b. Available in: <<https://www.blockcerts.org/guide/faq.html>>. Accessed on: 16 Jan. 2020.

BLOCKCHAIN. Average Number of Transactions Per Block. Available in: <<https://www.blockchain.com/charts/n-transactions-per-block?timespan=all>>. Accessed on: 13 Mar. 2020.

BLOCKCYPHER. BlockCypher's API documentation. Available in: <<https://www.blockcypher.com/dev/bitcoin/#introduction>>. Accessed on: 04 Dec. 2019.

BRASIL. **Constituição**: República Federativa do Brasil. Brasília, DF: Senado Federal. 1988. Available in:

<http://www.planalto.gov.br/ccivil_03/constituicao/constituicao.htm>. Accessed on: 15 Sep. 2019.

_____. **Medida Provisória No 2.200-2**. Brasília, DF: Senado Federal. 2001.

Available in: <http://www.planalto.gov.br/ccivil_03/mpv/antigas_2001/2200-2.htm>.

Accessed on: 15 Sep. 2019.

CERTISIGN. **Loja**. Available in: <<https://loja.certisign.com.br/home>>. Accessed on: 23 May. 2020.

COINTELEGRAPH. **What is Ethereum**. Available in:

<<https://cointelegraph.com/ethereum-for-beginners/what-is-ethereum>>. Accessed on: 22 Jan. 2020.

CRYSTAL. **Report On International Bitcoin Flows 2013 – 2019**. Available in:

<<https://crystalblockchain.com/assets/reports/International%20Bitcoin%20Flows%20Report%20for%202013-2019%20-%20by%20Crystal%20Blockchain,%20Bitfury.pdf>>. Accessed on: 18 Oct. 2019.

DAVIES, J. **Implementing SSL/TLS Using Cryptography and PKI**. Indianapolis, Indiana, USA: Wiley Publishing, 2011.

DEMOISELLE SIGNER. **Objetivos**. Available in:

<<https://www.frameworkdemoiselle.gov.br/v3/signer/>>. Accessed on: 13 Sep. 2019.

DIGITALOCEAN. **Droplets**. Available in:

<<https://www.digitalocean.com/docs/droplets/>>. Accessed on: 17 Sep. 2019.

DINIZ, E. **O blockchain veio para ficar**. GV-executivo, v. 17, n. 3, maio-junho, p.51, 2018. Available in: <https://rae.fgv.br/sites/rae.fgv.br/files/gv_v17n3_cl4.pdf>.

Accessed on: 31 May. 2020.

eSOCIAL. **Conheça o eSocial**. Available in:

<<http://portal.esocial.gov.br/institucional/conheca-o>>. Accessed on: 19 Jul. 2019.

ETERNITY WALL. Eternity Wall Android client. Version 1.0.28 [S.I.]: Eternity Wall, 04 Feb. 2016. Available in: <<https://github.com/RCasatta/EternityWallAndroid>>.

Accessed on: 29 Jun. 2020.

FERGUSON, N., SCHNEIER, B. **Practical Cryptography**. Indianapolis, Indiana, USA: Wiley Publishing, 2003.

FIPS. **What is FIPS**. Available in: <<https://whatis.techtarget.com/definition/FIPS-Federal-Information-Processing-Standards>>. Accessed on: 27 Aug. 2019.

FIPS PUB 180-4. **Secure Hash Standard (SHS)**. Available in:

<https://ws680.nist.gov/publication/get_pdf.cfm?pub_id=910977>. Accessed on: 27 Aug. 2019.

FURLAN, J. D. **Modelagem de Objetos Através da UML**. Campos Elíseos, São Paulo, Brazil: Makron Books, 1998.

G1. **Uso da Internet no Brasil Cresce, e 70% da População está Conectada**.

Available in: <<https://g1.globo.com/economia/tecnologia/noticia/2019/08/28/uso-da-internet-no-brasil-cresce-e-70percent-da-populacao-esta-conectada.ghtml>>.

Accessed on: 18 Oct. 2019.

GIL, A. C. **Como Elaborar Projetos de Pesquisa**. Campos Elíseos, São Paulo, Brazil: Atlas, 2017.

GLASSFISH. **The Open Source Java EE Reference Implementation**. Available in:

<<https://javaee.github.io/glassfish/download>>. Accessed on: 29 Jun. 2020.

HIGGINS, S. **Republic of Georgia to Develop Blockchain Land Registry.**

Available in: <<http://www.coindesk.com/bitfury-working-with-georgian-government-on-blockchain-land-registry/>>. Accessed on: 30 Jun. 2017

ID SEGURO. **e-Social:** certificado digital será obrigatório a partir de 1º de julho.

Available in: <<https://www.idseguro.com.br/calendario-e-social-certificado-digital-sera-obrigatorio-a-partir-de-1o-de-julho/>>. Accessed on: 19 Jul. 2019.

ITI. **Acordo de Reconhecimento Mútuo de Assinaturas Digitais no Mercosul.**

2020a. Available in: <<https://www.iti.gov.br/artigos/110-artigos-iti/4060-acordo-de-reconhecimento-mutuo-de-assinaturas-digitais-no-mercosul>>. Accessed on: 12 May. 2020.

____. **Benefícios.** 2020b. Available in: <<https://www.iti.gov.br/certificado-digital/2-uncategorised/95-beneficios>>. Accessed on: 12 May. 2020.

____. **Calendário e-Social:** certificado digital será obrigatório a partir de 1º de julho.

2020c. Available in: <<https://www.iti.gov.br/noticias/indice-de-noticias/2340-calendario-e-social-certificado-digital-sera-obrigatorio-a-partir-de-1-de-julho>>.

Accessed on: 12 May. 2020.

____. **Carimbo de Tempo.** 2020d. Available in: <<https://www.iti.gov.br/aceso-a-informacao/41-perguntas-frequentes/131-carimbo-do-tempo>>. Accessed on: 12 May.

2020.

____. **Entes da ICP-Brasil.** 2020e. Available in: <<https://www.iti.gov.br/icp-brasil/57-icp-%20brasil/76-como-funciona/>>. Accessed on: 12 May. 2020.

____. **Mercosul Avança na Digitalização e Países Reconhecerão Assinaturas**

Digitais mutuamente. 2020f. Available in: <<https://www.iti.gov.br/noticias-iti/4059-mercosul-avanca-na-digitalizacao-e-paises-reconhecerao-assinaturas-digitais-mutuamente>>. Accessed on: 12 May. 2020.

____. **O ITI**. 2020g. Available in: <<http://www.iti.gov.br/institucional/o-iti>>. Accessed on: 12 May. 2020.

____. **Ranking de Emissões**. 2020h. Available in: <<https://www.iti.gov.br/icp-brasil/100-ranking-de-emissoes>>. Accessed on: 12 May. 2020.

JONATHAN, K; YEHUDA, L. **Introduction to Modern Cryptography**. Boca Raton, Florida, USA: CHAPMAN & HALL CRC, 2015.

JÚNIOR, W. P; PEREIRA, V. L. D. V; FILHO, H. V. P. **Pesquisa Científica Sem Tropeços: Abordagem Sistêmica**. São Paulo, São Paulo, Brazil: Atlas, 2007.

KARAME, G.; ANDROULAKI, E. **Bitcoin and Blockchain Security**. Norwood, Massachusetts, USA: ARTECH HOUSE, 2016.

MINISTÉRIO DA SAÚDE. **Portaria Nº 467, de 20 de MARÇO de 2020**. Available in: <http://www.planalto.gov.br/CCIVIL_03/Portaria/PRT/Portaria%20n%C2%BA%20467-20-ms.htm>. Accessed on: 21 May. 2020.

NAKAMOTO, S. **Bitcoin: A Peer-to-Peer Electronic Cash System**. 2008. Available in: <<https://nakamotoinstitute.org/bitcoin>>. Accessed on: 23 Sep. 2019.

NOFER, M. et al. **Blockchain**. Bus Inf Syst Eng, vol. 59, n.3, p.183–187, 2017.

OBJECT MANAGEMENT GROUP. **What is UML**. Available in: <<https://www.uml.org/what-is-uml.htm>>. Accessed on: 17 Sep. 2019.

ORIGINALMY. **Blockchains e Verificação de Registros**. Available in: <https://originalmy.readthedocs.io/pt_BR/latest/60-blockchains.html>. Accessed on: 04 Feb. 2020

PROOF OF EXISTENCE. **What is proof of existence?** Accessed on: <<https://poex.io/about>>. Accessed on: 25 Oct. 2017.

RECEITA FEDERAL. **Relatórios do Resultado da Arrecadação**. Available in: <<http://idg.receita.fazenda.gov.br/dados/receitadata/arrecadacao/relatorios-do-resultado-da-arrecadacao>>. Accessed on: 14 Oct. 2019.

SAFEWEB. **Certificado Digital**. Available in: <<https://www2.safeweb.com.br/certificacao/certificacao.aspx>>. Accessed on: 16 Jan. 2019.

SCHOLLMEIER, R. **A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications**. Proceedings First International Conference on Peer-to-Peer Computing, Linkoping, Sweden, 2001, pp. 101-102.

SEARCHCLOUDCOMPUTING. **DigitalOcean**. Available in: <<https://searchcloudcomputing.techtarget.com/definition/DigitalOcean>>. Accessed on: 18 Sep. 2019.

SINGHAL, B.; DHAMEJA, G.; PANDA, P. S. **Beginning Blockchain: A Beginner's Guide to Building Blockchain Solutions**. New York, New York, USA: Apress Media, 2018.

SMERKIS, V. **Georgia Records 100,000 Land Titles on Bitcoin Blockchain: BitFury**. Available in: <<https://cointelegraph.com/news/georgia-records-100000-land-titles-on-bitcoin-blockchain-bitfury>>. Accessed on: 13 Jul. 2018.

STALLINGS, W. **Cryptography and network security**. Boston, Massachusetts, USA: Prentice Hall, 2011.

STATISTA. **How Common is Crypto?** Available in: <<https://www.statista.com/chart/18345/crypto-currency-adoption/>>. Accessed on: 18 Oct. 2018.

STINSON, D. **Cryptography: Theory and Practice**. Boca Banton, Florida, USA: CRC Press, 1995.

STRAY, K. **Non-Financial Blockchains - Where to Find Them**. Available in: <<https://cointelegraph.com/news/non-financial-blockchains-where-to-find-them>>. Accessed on: 13 Jul. 2019.

SWAN, M. **Blockchain: Blueprint for a New Economy**. Sebastopol, CA, USA: O'Reilly Media, 2015.

TSCHORSCH, F.; SCHEUERMANN, B. **Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies**. IEEE Communications Surveys & Tutorials, [s. l.], ano 3, v. 18, p. 2084 - 2123, 2016. DOI 10.1109/COMST.2016.2535718. Available in: <<https://ieeexplore.ieee.org/document/7423672>> . Accessed on: 28 Nov. 2019.

UNIVERSIDADE FEDERAL DO PARANÁ. **Gestão da Informação: Áreas de Concentração e Linhas de Pesquisa**. Available in: <<http://www.prppg.ufpr.br/site/ppggi/pb/linhas-de-pesquisa/>>. Accessed on: 22 May. 2020.

UNIX TIMESTAMP. **Timestamp Converter**. Available in: <<https://www.unixtimestamp.com/>>. Accessed on: 12 Mar. 2020.

VALID CERTIFICADORA BLOG. **VALID Certificadora lança assinatura digital para usuários de Blockchain**. Available in: <<https://blog.validcertificadora.com.br/valid-certificadora-lanca-assinatura-digital-para-usuarios-de-blockchain/>>. Accessed on: 17 Feb. 2020.

WHITMAN, M. E; MATTORD, H. J. **Principles of Information Security**. Boston, MA, USA: Cengage Learning, 2016.

WORLD ECONOMIC FORUM. **Center for the Fourth Industrial Revolution**. Available in: <<https://www.weforum.org/centre-for-the-fourth-industrial-revolution/areas-of-focus>>. Accessed on: 24 Oct. 2017.

ZHENG, Z. et al. **An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends.** In: proceeding of IEEE 6th International Congress on Big Data, 2017.

APPENDIX 1 – MAINACTIVITY OF ANDROID APP

```

1  package a.apkt;
2
3  import android.app.Activity;
4  import android.app.AlertDialog;
5  import android.app.Fragment;
6  import android.app.FragmentTransaction;
7  import android.content.DialogInterface;
8  import android.content.Intent;
9  import android.database.Cursor;
10 import android.graphics.drawable.ColorDrawable;
11 import android.net.Uri;
12 import android.os.Bundle;
13 import android.provider.OpenableColumns;
14 import android.support.v4.widget.DrawerLayout;
15 import android.support.v7.app.ActionBar;
16 import android.support.v7.app.AppCompatActivity;
17 import android.view.Gravity;
18 import android.view.LayoutInflater;
19 import android.view.Menu;
20 import android.view.MenuItem;
21 import android.view.View;
22 import android.widget.FrameLayout;
23 import android.widget.Spinner;
24 import android.widget.TextView;
25 import android.widget.Toast;
26
27 import com.google.gson.Gson;
28
29 import org.apache.log4j.chainsaw.Main;
30
31 import java.io.File;
32 import java.io.FileNotFoundException;
33 import java.io.IOException;
34 import java.io.InputStream;
35 import java.security.Key;
36 import java.security.KeyStore;
37 import java.security.PrivateKey;
38 import java.util.ArrayList;
39 import java.util.Iterator;
40 import java.util.List;
41
42 import a.apkt.asyncTask.OpReturnAsyncTask;
43 //import a.apkt.asyncTask.SignerAsyncTask;
44 import a.apkt.asyncTask.SignerAsyncTask;
45 import a.apkt.asyncTask.VerifyNotarizationAsyncTask;
46 import a.apkt.backingbean.LoginBB;
47 import a.apkt.json.ListDigCertJson;
48 import a.apkt.json.ListOpReturnJson;
49 import a.apkt.json.LoginJson;
50 import a.apkt.model.DigCert;
51 import a.apkt.model.OpReturn.OpReturnType;
52 import a.apkt.model.OpReturn;
53 import a.apkt.model.Tx;
54 import a.apkt.model.TxOutput;
55 import a.apkt.service.CheckConnectivity;
56 import a.apkt.service.StaticVars;
57 import a.apkt.service.UserMsgService;
58 import a.apkt.service.UtilsService;
59 import a.apkt.signer.KeyStorePKCS12;
60 import a.apkt.sqlite.LoginSqlite;
61
62 public class MainActivity extends AppCompatActivity

```

```

63     implements NavigationDrawerFragment.NavigationDrawerCallbacks,
64     ItemFragment.OnItemFragmentInteractionListener{
65
66     /**
67     * Fragment managing the behaviors, interactions and presentation of the navigation drawer.
68     */
69     private NavigationDrawerFragment mNavigationDrawerFragment;
70
71     /**
72     * Used to store the last screen title. For use in {@link #restoreActionBar()}.
73     */
74     private CharSequence mTitle;
75
76     private ActionBar actionBar;
77     private MenuItem actionAddItem;
78     private MenuItem actionImportItem;
79     private MenuItem actionPostItem;
80     private MenuItem actionWriteItem;
81     private MenuItem actionVerifyItem;
82
83     private long paramLoginId;
84     private Activity activity;
85     private String loginStringJson;
86     private LoginJson loginJson;
87     private ListOpReturnJson listOpReturnJson= new ListOpReturnJson();
88     private boolean isLoggedInin; //PUBLIC LIST CODE
89
90     private String selectedEmailAccount = null;
91     private Tx verifyNotarizationTx;
92     private TextView mEmailView;
93
94     private TextView mPwdView;
95     private String pwd = null;
96     private Uri uri;
97
98     private Fragment newFragment;
99
100    @Override
101    protected void onCreate(Bundle savedInstanceState) {
102        super.onCreate(savedInstanceState);
103        setContentView(R.layout.activity_main);
104        activity = this;
105
106        mNavigationDrawerFragment = (NavigationDrawerFragment)
107            getSupportFragmentManager().findFragmentById(R.id.navigation_drawer);
108        mTitle = getTitle();
109
110        // Set up the drawer.
111        mNavigationDrawerFragment.setUp(
112            R.id.navigation_drawer,
113            (DrawerLayout) findViewById(R.id.drawer_layout));
114    }
115
116    public void loadLoginJson() {
117        if (loginJson != null){
118            paramLoginId = loginJson.getId();
119            LoginBB loginBB = new LoginBB(loginJson);
120            loginStringJson = new Gson().toJson(loginBB.getLoginAux());
121            isLoggedInin = true;
122        }
123    }
124
125    @Override
126    public void onNavigationDrawerItemSelected(int position) {
127        // update the main content by replacing fragments
128        /*FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
129        FragmentManager fragmentManager = getSupportFragmentManager();
130        fragmentManager.beginTransaction()
131            .replace(R.id.container, PlaceholderFragment.newInstance(position + 1))
132            .commit();*/
133        if (isLoggedInin){
134            if (
135                position == StaticVars.TITLE_SECTION_WALL - 1
136                || position == StaticVars.TITLE_SECTION_NOTARIZE - 1
137                || position == StaticVars.TITLE_SECTION_DIGITAL_SIGNATURE - 1
138                || position == StaticVars.TITLE_SECTION_DIGITAL_CERTIFICATE - 1

```

```

139         ){
140         initItemFragment(position);
141     } else if (
142         position == StaticVars.TITLE_SECTION_VERIFY_NOTARIZATION - 1
143         ){
144         initVerifyNotarizationFragment(position);
145     } else if (
146         position == StaticVars.TITLE_SECTION_EXIT -1 /*Identification*/
147         ){
148         exit();
149     }
150
151 } else if (!isLoggedIn) {
152     if (
153         position == StaticVars.TITLE_SECTION_WALL_PUBLIC - 1
154         || position == StaticVars.TITLE_SECTION_NOTARIZE_PUBLIC - 1
155         // || position == StaticVars.TITLE_SECTION_DIGITAL_SIGNATURE_PUBLIC - 1
156         // || position == StaticVars.TITLE_SECTION_DIGITAL_CERTIFICATE_PUBLIC - 1
157         ){
158         initItemFragment(position);
159     } else if (
160         position == StaticVars.TITLE_SECTION_VERIFY_NOTARIZATION_PUBLIC - 1
161         ){
162         initVerifyNotarizationFragment(position);
163     }
164     // To activate Login search keyword ACTIVATELOGIN: uncomment code section containing
165     TITLE_SECTION_LOGIN
166     else if (
167         position == StaticVars.TITLE_SECTION_LOGIN - 1
168         ){
169         Intent it = new Intent(activity, LoginActivity.class);
170         Bundle params = new Bundle();
171         params.putBoolean("isLogin", true);
172         it.putExtras(params);
173         startActivity(it);
174         finish();
175     }
176 }
177
178 public void initItemFragment(int position) {
179     // Create new fragment and transaction
180     newFragment = new ItemFragment();
181
182     initFragment(position, newFragment);
183 }
184
185 public void initVerifyNotarizationFragment(int position) {
186     // Create new fragment and transaction
187     Fragment newFragment = new VerifyNotarizationFragment();
188
189     initFragment(position, newFragment);
190 }
191
192 public void initFragment(int position, Fragment newFragment) {
193     Intent itLogin = getIntent();
194     if (itLogin != null) {
195         Bundle params = itLogin.getExtras();
196         if (params != null && loginJson == null) {
197             loginJson = new Gson().fromJson(params.getString("loginStringJson"), LoginJson.class);
198         }
199     }
200
201     loadLoginJson();
202
203     Bundle args = new Bundle();
204     args.putInt("section_number", position + 1);
205     if (loginStringJson != null){
206         args.putString("loginStringJson", loginStringJson);
207     }
208
209     newFragment.setArguments(args);
210     FragmentTransaction transaction = getFragmentManager().beginTransaction();
211
212     // Replace whatever is in the fragment_container view with this fragment,
213     // and add the transaction to the back stack

```

```

214     transaction.replace(R.id.container, newFragment);
215
216     // Commit the transaction
217     transaction.commit();
218 }
219
220 public void exit(){
221     LoginSqlite loginSqlite = new LoginSqlite(this);
222     loginSqlite.deleteUserLogin();
223     finish();
224 }
225
226 public void onSectionAttached(int number) {
227     if (isLoggedIn){
228         switch (number) {
229             case StaticVars.TITLE_SECTION_DIGITAL_SIGNATURE:
230                 mTitle = getString(R.string.title_section_digital_signatures);
231                 break;
232             case StaticVars.TITLE_SECTION_DIGITAL_CERTIFICATE:
233                 mTitle = getString(R.string.title_section_digital_certificates);
234                 break;
235             case StaticVars.TITLE_SECTION_WALL:
236                 mTitle = getString(R.string.title_section_wall);
237                 break;
238             case StaticVars.TITLE_SECTION_NOTARIZE:
239                 mTitle = getString(R.string.title_section_notarizations);
240                 break;
241             case StaticVars.TITLE_SECTION_VERIFY_NOTARIZATION:
242                 mTitle = getString(R.string.title_section_verify_notarization);
243                 break;
244         }
245     } else if (!isLoggedIn) {
246         switch (number) {
247             // case StaticVars.TITLE_SECTION_DIGITAL_SIGNATURE_PUBLIC:
248             //     mTitle = getString(R.string.title_section_digital_signatures);
249             //     break;
250             // case StaticVars.TITLE_SECTION_DIGITAL_CERTIFICATE_PUBLIC:
251             //     mTitle = getString(R.string.title_section_digital_certificates);
252             //     break;
253             case StaticVars.TITLE_SECTION_WALL_PUBLIC:
254                 mTitle = getString(R.string.title_section_wall);
255                 break;
256             case StaticVars.TITLE_SECTION_NOTARIZE_PUBLIC:
257                 mTitle = getString(R.string.title_section_notarizations);
258                 break;
259             case StaticVars.TITLE_SECTION_VERIFY_NOTARIZATION_PUBLIC:
260                 mTitle = getString(R.string.title_section_verify_notarization);
261                 break;
262             // To activate Login search keyword ACTIVATELOGIN: uncomment code section containing
263             // TITLE_SECTION_LOGIN
264             case StaticVars.TITLE_SECTION_LOGIN:
265                 mTitle = getString(R.string.title_section_login);
266                 break;
267         }
268     }
269
270     public void restoreActionBar() {
271         actionBar = getSupportActionBar();
272         actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_STANDARD);
273         actionBar.setDisplayHomeAsUpEnabled(true);
274         actionBar.setTitle(mTitle);
275
276         // code block to setup ic_drawer
277         actionBar.setHomeButtonEnabled(true);
278         actionBar.setDisplayHomeAsUpEnabled(true);
279         actionBar.setHomeAsUpIndicator(R.drawable.ic_drawer_white);
280         // end of code block to setup ic_drawer
281     }
282
283     @Override
284     public boolean onCreateOptionsMenu(Menu menu) {
285         if (!mNavigationDrawerFragment.isDrawerOpen()) {
286             // Only show items in the action bar relevant to this screen
287             // if the drawer is not showing. Otherwise, let the drawer
288             // decide what to show in the action bar.

```

```

289     getMenuInflater().inflate(R.menu.main, menu);
290     actionAddItem = menu.findItem(R.id.action_add);
291     actionImportItem = menu.findItem(R.id.action_import);
292     actionPostItem = menu.findItem(R.id.action_post);
293     actionWriteItem = menu.findItem(R.id.action_write);
294     actionVerifyItem = menu.findItem(R.id.action_verify);
295     restoreActionBar();
296     setActionItemVisible();
297     return true;
298 }
299 return super.onCreateOptionsMenu(menu);
300 }
301
302 public void setActionItemVisible() {
303     if (
304         mTitle.equals(getString(R.string.title_section_digital_signatures))
305     ){
306         actionBar.setBackgroundDrawable(new ColorDrawable(getResources().getColor(R.color.blue_actionbar)));
307         actionAddItem.setVisible(true);
308         actionImportItem.setVisible(false);
309         actionPostItem.setVisible(false);
310         actionWriteItem.setVisible(false);
311         actionVerifyItem.setVisible(false);
312     } else if (
313         mTitle.equals(getString(R.string.title_section_digital_certificates))
314     ){
315         actionBar.setBackgroundDrawable(new ColorDrawable(getResources().getColor(R.color.blue_actionbar)));
316         actionAddItem.setVisible(false);
317         actionImportItem.setVisible(true);
318         actionPostItem.setVisible(false);
319         actionWriteItem.setVisible(false);
320         actionVerifyItem.setVisible(false);
321     } else if (
322         mTitle.equals(getString(R.string.title_section_wall))
323     ){
324         actionBar.setBackgroundDrawable(new ColorDrawable(getResources().getColor(R.color.blue_actionbar)));
325         actionAddItem.setVisible(false);
326         actionImportItem.setVisible(false);
327         actionPostItem.setVisible(false);
328         actionWriteItem.setVisible(true);
329         actionVerifyItem.setVisible(false);
330     } else if (
331         mTitle.equals(getString(R.string.title_section_notarizations))
332     ){
333         actionBar.setBackgroundDrawable(new ColorDrawable(getResources().getColor(R.color.blue_actionbar)));
334         actionAddItem.setVisible(true);
335         actionImportItem.setVisible(false);
336         actionPostItem.setVisible(false);
337         actionWriteItem.setVisible(false);
338         actionVerifyItem.setVisible(false);
339     } else if (
340         mTitle.equals(getString(R.string.title_section_verify_notarization))
341     ){
342         actionBar.setBackgroundDrawable(new ColorDrawable(getResources().getColor(R.color.blue_actionbar)));
343         actionAddItem.setVisible(true);
344         actionImportItem.setVisible(false);
345         actionPostItem.setVisible(false);
346         actionWriteItem.setVisible(false);
347         actionVerifyItem.setVisible(false);
348     }
349 }
350
351 @Override
352 public boolean onOptionsItemSelected(MenuItem item) {
353     // Handle action bar item clicks here. The action bar will
354     // automatically handle clicks on the Home/Up button, so long
355     // as you specify a parent activity in AndroidManifest.xml.
356     int id = item.getItemId();
357
358     if (actionAddItem != null && id == actionAddItem.getItemId()) {
359         if (mTitle.toString().equals(getString(R.string.title_section_digital_signatures))) {
360             Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
361             intent.setType("*/*");
362             startActivityForResult(intent, StaticVars.OPEN_DOCUMENT_SIGNATURE_ACTIVITY_RESULT);
363             return true;
364         } else

```

```

365     if (mTitle.toString().equals(getString(R.string.title_section_notarizations))) {
366
367         // add a view to AlertDialog
368         FrameLayout frameView = new FrameLayout(activity);
369         LayoutInflater inflater = activity.getLayoutInflater();
370         View dialoglayout = inflater.inflate(R.layout.layout_email,
371             frameView);
372
373         mEmailView = (TextView) dialoglayout
374             .findViewById(R.id.email);
375         Spinner mEmailSpinner = (Spinner) dialoglayout
376             .findViewById(R.id.email_spinner);
377
378         mEmailSpinner.setVisibility(View.GONE);
379
380         // keyword search to undo this conde change: GET_EMAIL_WITH_SPINNER. email spinner not used because
381         // AndroidManifest permission GET_ACCOUNTS requires privacy policy in order to publish on Google Play
382         emailAccounts = UtilsService.getRegisteredEmailAddresses(activity);
383         if (emailAccounts == null) {
384             mEmailSpinner.setVisibility(View.GONE);
385             mEmailView.setEnabled(false);
386             UserMsgService.showDialog(activity,
387                 R.string.error_email_nonexistent_title,
388                 R.string.error_email_nonexistent);
389         } else if (emailAccounts.size() == 1) {
390             mEmailSpinner.setVisibility(View.GONE);
391
392             selectedEmailAccount = emailAccounts.get(0);
393
394             mEmailView.setText(selectedEmailAccount);
395         } else if (emailAccounts.size() >= 2) {
396
397             mEmailView.setVisibility(View.GONE);
398
399             // Create an ArrayAdapter using the string array and a default
400             // spinner layout
401             ArrayAdapter<String> adapter = new ArrayAdapter<>(
402                 activity, R.layout.spinner_text_black_color,
403                 emailAccounts);
404             // Specify the layout to use when the list of choices appears
405             adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
406             // Apply the adapter to the spinner
407             mEmailSpinner.setAdapter(adapter);
408             mEmailSpinner
409                 .setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
410
411                 @Override
412                 public void onItemSelected(AdapterView<?> arg0,
413                     View arg1, int positon, long arg3) {
414                     selectedEmailAccount = emailAccounts
415                         .get(positon);
416                 }
417
418                 @Override
419                 public void onNothingSelected(AdapterView<?> arg0) {
420                     // TODO Auto-generated method stub
421                 }
422             });
423         }
424
425         new AlertDialog.Builder(activity)
426             .setTitle(
427                 R.string.alertdialog_email_address_title)
428             .setMessage(
429                 R.string.alertdialog_email_address)
430             .setView(frameView)
431             .setPositiveButton(getText(R.string.ok).toString(),
432                 new DialogInterface.OnClickListener() {
433                     public void onClick(DialogInterface dialog,
434                         int whichButton) {
435                         Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT); // (ETERNITY WALL, 2016)
436                         intent.setType("*/"); // (ETERNITY WALL, 2016)
437                         startActivityForResult(intent, StaticVars.OPEN_DOCUMENT_NOTARIZE_ACTIVITY_RESULT);
438

```

```

439
440 //
441 //
442 //         .setNegativeButton(getText(R.string.action_cancel).toString(),
443 //             new DialogInterface.OnClickListener() {
444 //                 public void onClick(DialogInterface dialog,
445 //                     int whichButton) {
446 //                     dialog.dismiss();
447 //                 }
448 //             }).show();
449
450     return true;
451 } else if (mTitle.toString().equals(getString(R.string.title_section_verify_notarization))) {
452     VerifyNotarizationFragment verifyNotarizationFragment =
453         (VerifyNotarizationFragment) fragmentManager.findFragmentById((R.id.container));
454
455     String txId = verifyNotarizationFragment.getTxIdFromVerifyNotarizationFragment();
456     if (CheckConnectivity.checkNow(this)) {
457         VerifyNotarizationAsyncTask verifyNotarizationAsyncTask = new VerifyNotarizationAsyncTask(activity,
458             actionAddItem, txId);
459         verifyNotarizationAsyncTask.execute();
460     } else {
461         UserMsgService.showDialog(this, R.string.alertdialog_no_connectivity_title,
462             R.string.alertdialog_no_connectivity);
463     }
464     return true;
465 }
466 } else if (actionImportItem != null && id == actionImportItem.getItemId()) {
467     Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT); // (ETERNITY WALL, 2016)
468     intent.setType("*/**"); // (ETERNITY WALL, 2016)
469     startActivityForResult(intent, StaticVars.OPEN_DOCUMENT_DIGITAL_CERTIFICATE_ACTIVITY_RESULT);
470     return true;
471 } else if (actionWriteItem != null && id == actionWriteItem.getItemId()){
472
473     // add a view to AlertDialog
474     FrameLayout frameView = new FrameLayout(activity);
475     LayoutInflater inflater = activity.getLayoutInflater();
476     View dialoglayout = inflater.inflate(R.layout.layout_email,
477         frameView);
478
479     mEmailView = (TextView) dialoglayout
480         .findViewById(R.id.email);
481     Spinner mEmailSpinner = (Spinner) dialoglayout
482         .findViewById(R.id.email_spinner);
483
484     mEmailSpinner.setVisibility(View.GONE);
485
486     new AlertDialog.Builder(activity)
487         .setTitle(
488             R.string.alertdialog_email_address_title)
489         .setMessage(
490             R.string.alertdialog_email_address)
491         .setView(frameView)
492         .setPositiveButton(getText(R.string.ok).toString(),
493             new DialogInterface.OnClickListener() {
494                 public void onClick(DialogInterface dialog,
495                     int whichButton) {
496                     Intent intent = new Intent(activity, OpReturnActivity.class);
497                     Bundle params = new Bundle();
498
499                     if (loginJson == null){
500                         loginJson = new LoginJson();
501                     }
502
503                     // keyword search to undo this code change: GET_EMAIL_WITH_SPINNER.
504                     // Get selectedEmailAccount from EmailView because email spinner is not used since
505                     AndroidManifest.permission GET_ACCOUNTS requires privacy policy in order to publish on Google Play
506                     // delete or comment code below when email spinner is used
507                     selectedEmailAccount = mEmailView.getText().toString();
508                     loginJson.setEmail(selectedEmailAccount);
509                     LoginBB loginBB = new LoginBB(loginJson);
510                     loginStringJson = new Gson().toJson(loginBB.getLoginAux());
511                     loginJson = null;
512                     params.putString("loginStringJson", loginStringJson);
513                     intent.putExtras(params);
514                     if (isLoggedIn){

```



```

512         startActivityForResult(intent, StaticVars.OP_RETURN_ACTIVITY_RESULT);
513     } else {
514         startActivityForResult(intent, StaticVars.OP_RETURN_ACTIVITY_PUBLIC_RESULT);
515     }
516 }
517 })
518 .setNegativeButton(getText(R.string.action_cancel).toString(),
519     new DialogInterface.OnClickListener() {
520         public void onClick(DialogInterface dialog,
521             int whichButton) {
522             dialog.dismiss();
523         }
524     }).show();
525     return true;
526 }
527 // else if (id == R.id.action_settings) {
528 //     return true;
529 // }
530
531     return super.onOptionsItemSelected(item);
532 }
533
534 @Override
535 protected void onActivityResult(int requestCode, int resultCode, Intent data) {
536     super.onActivityResult(requestCode, resultCode, data);
537
538     if (requestCode == StaticVars.OPEN_DOCUMENT_SIGNATURE_ACTIVITY_RESULT) {
539         if (data != null && data.getData() != null) {
540             uri = data.getData(); // (ETERNITY WALL, 2016)
541
542             // add a view to AlertDialog
543             FrameLayout frameView = new FrameLayout(activity);
544             LayoutInflater inflater = activity.getLayoutInflater();
545             View dialoglayout = inflater.inflate(R.layout.layout_password,
546                 frameView);
547
548             mPwdView = (TextView) dialoglayout
549                 .findViewById(R.id.password);
550
551             new AlertDialog.Builder(activity)
552                 .setTitle(
553                     R.string.alertdialog_import_digital_certificate_title)
554                 .setMessage(
555                     R.string.alertdialog_import_digital_certificate)
556                 .setView(frameView)
557                 .setPositiveButton(getText(R.string.ok).toString(),
558                     new DialogInterface.OnClickListener() {
559                         public void onClick(DialogInterface dialog,
560                             int whichButton) {
561                             MainActivity mainActivity = (MainActivity) activity;
562                             pwd = mPwdView.getText().toString();
563                             SignerAsyncTask signerAsyncTask = new SignerAsyncTask(null, null, mainActivity, uri, pwd);
564                             signerAsyncTask.execute();
565                             uri = null;
566                         }
567                     })
568                 .setNegativeButton(getText(R.string.action_cancel).toString(),
569                     new DialogInterface.OnClickListener() {
570                         public void onClick(DialogInterface dialog,
571                             int whichButton) {
572                             dialog.dismiss();
573                         }
574                     }).show();
575
576         }
577     } else
578     // import digital certificate
579     if (requestCode == StaticVars.OPEN_DOCUMENT_DIGITAL_CERTIFICATE_ACTIVITY_RESULT) {
580         if (data != null && data.getData() != null) {
581             // get digital certificate path
582             uri = data.getData(); // (ETERNITY WALL, 2016)
583
584             // loads user interface
585             FrameLayout frameView = new FrameLayout(activity);
586             LayoutInflater inflater = activity.getLayoutInflater();

```



```

661         DigCert digCertIt = (DigCert) iterator.next();
662         InputStream digCertItStream =
UtilsService.convertBytesToInputStream(digCertIt.getDigCertBytes());
663         String digCertSha256Iterator = UtilsService.sha256Doc(activity, digCertItStream);
664         if (digCertSha256.equals(digCertSha256Iterator)) {
665             existDigCert = true;
666         }
667     }
668 }
669 // save digital certificate from uri path in internal storage if verified that
670 // was not yet stored in internal storage
671 if (!existDigCert) {
672
673     DigCert digCert = KeyStorePKCS12.certInfo(KeyStorePKCS12.certificateChain(keyStore));
674     digCert.setFileName(certName);
675     digCert.setDigCertBytes(digCertBytes);
676     digCertList.add(digCert);
677     UtilsService.saveObjectInternalStorage(activity, digCertList, StaticVars.DIGCERTLIST);
678 }
679 initItemFragment(StaticVars.TITLE_SECTION_DIGITAL_CERTIFICATE - 1);
680 uri = null;
681 } catch (FileNotFoundException e) {
682     e.printStackTrace();
683     uri = null;
684 } catch (IOException e) {
685     e.printStackTrace();
686     uri = null;
687 } catch (Exception e) {
688     e.printStackTrace();
689     uri = null;
690 }
691 }
692 })
693 .setNegativeButton(getText(R.string.action_cancel).toString(),
694     new DialogInterface.OnClickListener() {
695         public void onClick(DialogInterface dialog,
696             int whichButton) {
697             dialog.dismiss();
698         }
699     }).show();
700
701 } else {
702     Toast toast = Toast.makeText(activity, activity.getString(R.string.toast_file_invalid), Toast.LENGTH_LONG);
703     toast.setGravity(Gravity.BOTTOM, 0, 0);
704     toast.show();
705 }
706 }
707 }
708 }
709 else if (requestCode == StaticVars.OP_RETURN_ACTIVITY_RESULT) {
710     // Make sure the request was successful
711     if (resultCode == RESULT_OK) {
712         initItemFragment(StaticVars.TITLE_SECTION_WALL - 1);
713     }
714 }
715 }
716 else if (requestCode == StaticVars.OP_RETURN_ACTIVITY_PUBLIC_RESULT) {
717     // Make sure the request was successful
718     if (resultCode == RESULT_OK) {
719         initItemFragment(StaticVars.TITLE_SECTION_WALL_PUBLIC - 1);
720     }
721 }
722 }
723 else if (requestCode == StaticVars.NOTARIZE_RESULT) {
724     // Make sure the request was successful
725     if (resultCode == RESULT_OK) {
726         initItemFragment(StaticVars.TITLE_SECTION_NOTARIZE - 1);
727     }
728 }
729 }
730 else if (requestCode == StaticVars.NOTARIZE_PUBLIC_RESULT) {
731     // Make sure the request was successful
732     if (resultCode == RESULT_OK) {
733         initItemFragment(StaticVars.TITLE_SECTION_NOTARIZE_PUBLIC - 1);
734     }
735 }

```

```

736     }
737     else if (requestCode == StaticVars.OPEN_DOCUMENT_NOTARIZE_ACTIVITY_RESULT) {
738         if (data != null && data.getData() != null) {
739             Uri uri = data.getData(); // (ETERNITY WALL, 2016)
740             String path =uri.toString();
741             InputStream is = null;
742             try {
743                 is = activity.getContentResolver().openInputStream(uri);
744             } catch (FileNotFoundException e) {
745                 e.printStackTrace();
746             }
747             String dataHexHashString = UtilsService.sha256Doc(activity, is);
748
749             // run
750             if(path!=null && dataHexHashString!=null){
751                 Boolean conn = CheckConnectivity.checkNow(activity);
752                 if (conn) {
753
754                     // keyword search to undo this code change: GET_EMAIL_WITH_SPINNER.
755                     // Get selectedEmailAccount from EmailView because email spinner is not used since AndroidManifest
756                     // permission GET_ACCOUNTS requires privacy policy in order to publish on Google Play
757                     // delete or comment code below when email spinner is used
758                     selectedEmailAccount = mEmailView.getText().toString();
759
760                     OpReturnAsyncTask opReturnAsyncTask = new OpReturnAsyncTask(
761                         null, // View
762                         null, // ScrollView
763                         activity,
764                         loginJson,
765                         dataHexHashString,
766                         selectedEmailAccount,
767                         OpReturnType.OP_RETURN_TYPE_NOTARIZATION);
768                     opReturnAsyncTask.execute();
769
770                 } else {
771                     UserMsgService.showDialog(activity, R.string.alertdialog_no_connectivity_title,
772                         R.string.alertdialog_no_connectivity);
773                 }
774             }
775     } else if (requestCode == StaticVars.OPEN_DOCUMENT_VERIFY_NOTARIZATION_ACTIVITY_RESULT) {
776         if (data != null && data.getData() != null) {
777             Uri uri = data.getData(); // (ETERNITY WALL, 2016)
778             InputStream is = null;
779             try {
780                 is = activity.getContentResolver().openInputStream(uri);
781             } catch (FileNotFoundException e) {
782                 e.printStackTrace();
783             }
784             String dataHexHashString = UtilsService.sha256Doc(activity, is);
785
786             if (dataHexHashString != null){
787                 VerifyNotarizationFragment verifyNotarizationFragment =
788                     (VerifyNotarizationFragment) getFragmentManager().findFragmentById((R.id.container));
789                 // set EditText et_tx_id to null, so that when user returns to fragment the editText will be empty
790                 verifyNotarizationFragment.setTxIdFromVerifyNotarizationFragment();
791                 boolean isDataHexEqual = false;
792                 for (TxOutput to : verifyNotarizationTx.getOutputs()){
793                     String dataHex= to.getDataHex();
794                     if (dataHexHashString.equals(dataHex)){
795                         isDataHexEqual = true;
796                     }
797                 }
798
799                 Intent intent = new Intent(activity, VerifyNotarizationActivity.class);
800                 Bundle args = new Bundle();
801                 String verifyNotarizationTxJson = new Gson().toJson(verifyNotarizationTx);
802                 args.putString("verifyNotarizationTxJson", verifyNotarizationTxJson);
803                 args.putString("dataHexHashString", dataHexHashString);
804                 args.putBoolean("isDataHexEqual", isDataHexEqual);
805                 intent.putExtras(args);
806                 activity.startActivity(intent);
807             }
808         }
809     } else if (requestCode == StaticVars.SELECT_BITCOIN_WALLET_ACTIVITY_RESULT) {

```

```

810     // Make sure the request was successful
811     if (resultCode == RESULT_OK // RESULT_CANCELED works with testnet wallet on testnet network
812         || resultCode == RESULT_CANCELED // RESULT_CANCELED works with AirBitz wallet and other wallets on
main network
813     ) {
814
815         Intent it = new Intent(activity, OpReturnRqstdActivity.class);
816         Bundle args = new Bundle();
817         args.putString("titleSection", getResources().getText(R.string.title_section_notarizations).toString());
818         it.putExtras(args);
819         if (isLoggedIn){
820             startActivityForResult(it, StaticVars.NOTARIZE_RESULT);
821         } else {
822             startActivityForResult(it, StaticVars.NOTARIZE_PUBLIC_RESULT);
823         }
824     }
825 }
826 } else if (requestCode == StaticVars.DIG_CERT_ACTIVITY_RESULT) {
827     // Make sure the request was successful
828     if (resultCode == RESULT_OK) {
829
830         initItemFragment(StaticVars.TITLE_SECTION_DIGITAL_CERTIFICATE- 1);
831     }
832 }
833 }
834
835 @Override
836 public List<OpReturn> onOpReturnItemFragmentInteraction() {
837     return listOpReturnJson.getOpReturnList();
838 }
839
840 @Override
841 public void onSetOpReturnItemFragmentInteraction(List<OpReturn> opReturnList) {
842     listOpReturnJson.setOpReturnList(opReturnList);
843 }
844
845 @Override
846 public String onActionBarSetTitle() {
847     if (loginJson != null){
848         return loginJson.getUsername();
849     } else {
850         return null;
851     }
852 }
853
854 //PUBLIC LIST CODE
855 @Override
856 public boolean onLogin() {
857     return isLoggedIn;
858 }
859
860 public void setVerifyNotarizationTx(Tx verifyNotarizationTx){
861     this.verifyNotarizationTx = verifyNotarizationTx;
862 }
863
864 }

```

APPENDIX 2 – SIGNERASYNC TASK OF ANDROID APP

```

1  package a.apkt.asyncTask;
2
3  import android.animation.Animator;
4  import android.animation.AnimatorListenerAdapter;
5  import android.annotation.TargetApi;
6  import android.app.Activity;
7  import android.database.CursorIndexOutOfBoundsException;
8  import android.net.Uri;
9  import android.os.AsyncTask;
10 import android.os.Build;
11 import android.os.Environment;
12 import android.view.View;
13 import android.widget.ScrollView;
14
15 import org.demoselle.signer.policy.impl.cades.pkcs7.PKCS7Signer;
16
17 import java.io.File;
18 import java.io.FileNotFoundException;
19 import java.io.FileOutputStream;
20 import java.io.IOException;
21 import java.io.InputStream;
22 import java.io.OutputStream;
23 import java.io.PrintWriter;
24 import java.net.ConnectException;
25 import java.security.KeyStore;
26 import java.security.PrivateKey;
27 import java.util.List;
28
29 import a.apkt.MainActivity;
30 import a.apkt.R;
31 import a.apkt.model.DigCert;
32 import a.apkt.service.StaticVars;
33 import a.apkt.service.UserMsgService;
34 import a.apkt.service.UtilsService;
35 import a.apkt.signer.KeyStorePKCS12;
36
37 public class SignerAsyncTask extends AsyncTask<Void, Void, Boolean> {
38     private boolean glassfishDown = false;
39     private MainActivity mainActivity;
40     private Uri uri;
41     private String pwd;
42     private View mProgressBar;
43     private ScrollView scrollView;
44
45     public SignerAsyncTask (
46         View mProgressBar,
47         ScrollView scrollView,
48         MainActivity mainActivity,
49         Uri uri,
50         String pwd){
51         this.mainActivity = mainActivity;
52         this.uri = uri;
53         this.pwd = pwd;
54     }
55
56     @Override
57     protected Boolean doInBackground(Void... voids) {
58

```

```

59 try {
60     // get list of digital certificates from internal storage
61     List<DigCert> digCertList = UtilsService.getDigCertList(mainActivity);
62
63     // get digital certificate in first position in the list, because as of now
64     // the app is coded to deal with only one digital certificate when signing documents
65     DigCert digCert = digCertList.get(0);
66
67     // get bytes format of digital certificate
68     byte[] digCertBytes = digCert.getDigCertBytes();
69
70     // convert bytes format of digital certificate into inputStream
71     InputStream digCertInputStream = UtilsService.convertBytesToInputStream(digCertBytes);
72
73     // get digital certificate keystore if password is correct; otherwise, throw exception
74     KeyStore keyStore = KeyStorePKCS12.loadKeystore(mainActivity, digCertInputStream, pwd);
75
76     // load private key from digital certificate
77     PrivateKey privateKey = KeyStorePKCS12.loadPrivKey(keyStore);
78
79     // get PKCS7 format for implementation of digital signatures
80     PKCS7Signer signer = KeyStorePKCS12.signer(keyStore, privateKey);
81
82     // load document from uri path into InputStream
83     InputStream inputStream = mainActivity.getContentResolver().openInputStream(uri);
84
85     // convert InputStream of document into bytes
86     byte[] fileBytes = UtilsService.convertInputStreamToBytes(inputStream);
87
88     // sign document and get bytes format of signed document
89     byte[] fileSignedBytes = signer.doDetachedSign(fileBytes);
90
91     // get name of signed document
92     String fileName = UtilsService.retrieveFileName(mainActivity, uri);
93
94     // create directory before attempting to a save file in a new directory
95     File directory = new File(Environment.getExternalStorageDirectory() + File.separator +
mainActivity.getString(R.string.app_name));
96     directory.mkdirs();
97
98     // save signed file to directory
99     String fileDirectorySignedPath = Environment.getExternalStorageDirectory() +
100     File.separator + mainActivity.getString(R.string.app_name) + File.separator + fileName +
"signedDettached_" + ".p7s";
101     UtilsService.savefileExternalStorage(fileSignedBytes, fileDirectorySignedPath);
102
103 } catch (ConnectException exception) {
104
105     glassfishDown = true;
106     return false;
107
108 } catch (CursorIndexOutOfBoundsException e) {
109     e.getMessage();
110 }
111 catch (Exception e) {
112     e.getMessage();
113     glassfishDown = true;
114     return false;
115 }
116
117 // List<String> certInfoList = KeyStorePKCS12.certInfo(KeyStorePKCS12.certificateChain(keyStore));
118 //
119 // PrivateKey chavePrivada = KeyStorePKCS12.loadPrivKey(keyStore);
120 //
121 // PKCS7Signer signer = KeyStorePKCS12.signer(keyStore, chavePrivada);
122 //
123 // // code block for signing docs
124 // //*****
125 // try {
126 //
127 //     File fileSign = new File(Environment.getExternalStorageDirectory() +
128 //     File.separator + Environment.DIRECTORY_DOWNLOADS + File.separator + "encryption.jpg");
129 //     Uri uriFileSign = Uri.fromFile(fileSign);
130 //     InputStream is = activity.getContentResolver().openInputStream(uriFileSign);
131 //     int fileSize = is.available();
132 //     byte[] resultFileSign = new byte[fileSize];

```

```

133 //     is.read(resultFileSign);
134 //     is.close();
135 //
136 //     // line of code needed in async task
137 //     byte[] signature = signer.doDetachedSign(resultFileSign);
138 //
139 //     OutputStream outFileSign = new FileOutputStream(Environment.getExternalStorageDirectory() +
140 //         File.separator + activity.getString(R.string.app_name) + File.separator + "DIRECTORY_DOWNLOADS.p7s");
141 //
142 //     outFileSign.write(signature);
143 //     outFileSign.close();
144 //
145 //     } catch (FileNotFoundException e) {
146 //         e.printStackTrace();
147 //     } catch (IOException e) {
148 //         e.printStackTrace();
149 //     }
150 // *****
151
152     return true;
153 }
154
155 @Override
156 protected void onPostExecute(final Boolean success) {
157
158     if (success) {
159         mainActivity.initItemFragment(StaticVars.TITLE_SECTION_DIGITAL_SIGNATURE - 1);
160     }
161     else if (!success) {
162         if (glassfishDown) {
163             UserMsgService.showDialogPositButtonMail(mainActivity,
164                 mainActivity, R.string.alertdialog_internal_problem_title,
165                 R.string.alertdialog_internal_problem_msg);
166         }
167     }
168 }
169
170 /**
171  * Shows the progress UI and hides the login form.
172  */
173 @TargetApi(Build.VERSION_CODES.HONEYCOMB_MR2)
174 public void showProgress(final boolean show) {
175
176     // On Honeycomb MR2 we have the ViewPropertyAnimator APIs, which allow
177     // for very easy animations. If available, use these APIs to fade-in
178     // the progress spinner.
179     if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB_MR2) {
180         int shortAnimTime = mainActivity.getResources().getInteger(android.R.integer.config_shortAnimTime);
181
182         mActivityFormView.setVisibility(show ? View.GONE : View.VISIBLE);
183         mActivityFormView.animate().setDuration(shortAnimTime).alpha(
184             show ? 0 : 1).setListener(new AnimatorListenerAdapter() {
185             @Override
186             public void onAnimationEnd(Animator animation) {
187                 mActivityFormView.setVisibility(show ? View.GONE : View.VISIBLE);
188             }
189         });
190
191         scrollView.setVisibility(show ? View.GONE : View.VISIBLE);
192         mProgressView.setVisibility(show ? View.VISIBLE : View.GONE);
193         mProgressView.animate().setDuration(shortAnimTime).alpha(
194             show ? 1 : 0).setListener(new AnimatorListenerAdapter() {
195             @Override
196             public void onAnimationEnd(Animator animation) {
197                 mProgressView.setVisibility(show ? View.VISIBLE : View.GONE);
198                 scrollView.setVisibility(show ? View.GONE : View.VISIBLE);
199             }
200         });
201     } else {
202         // The ViewPropertyAnimator APIs are not available, so simply show
203         // and hide the relevant UI components.
204         mProgressView.setVisibility(show ? View.VISIBLE : View.GONE);
205         scrollView.setVisibility(show ? View.GONE : View.VISIBLE);
206
207     }
208 }

```

209
210 }

APPENDIX 3 – OPRETURNMAIN OF JAVA WEB APPLICATION SERVER

```

1 /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6 package apkt.opreturn;
7
8 import apkt.dao.jpa.GenericDaoJpa;
9 import apkt.mail.JavaMailThread;
10 import apkt.model.OpReturn;
11 import apkt.model.OpReturn.OpReturnType;
12 import apkt.model.TxOpReturn;
13 import apkt.service.ProjService;
14 import com.google.common.util.concurrent.Service;
15 import java.io.File;
16 import java.io.IOException;
17 import java.math.BigDecimal;
18 import java.math.RoundingMode;
19 import java.net.URL;
20 import java.net.URLClassLoader;
21 import java.util.Date;
22 import java.util.Iterator;
23 import java.util.List;
24 import java.util.Locale;
25 import java.util.ResourceBundle;
26 import java.util.Set;
27 import java.util.concurrent.ExecutorService;
28 import java.util.concurrent.Executors;
29 import java.util.logging.Level;
30 import java.util.logging.Logger;
31 import javax.persistence.EntityManager;
32 import javax.persistence.EntityManagerFactory;
33 import javax.persistence.Persistence;
34 import org.bitcoinj.core.Address;
35 import org.bitcoinj.core.Coin;
36 import org.bitcoinj.core.InsufficientMoneyException;
37 import org.bitcoinj.core.NetworkParameters;
38 import org.bitcoinj.core.Transaction;
39 import org.bitcoinj.core.TransactionOutput;
40 import org.bitcoinj.kits.WalletAppKit;
41 import org.bitcoinj.params.MainNetParams;
42 import org.bitcoinj.params.TestNet3Params;
43 import org.bitcoinj.script.ScriptBuilder;
44 import org.bitcoinj.wallet.SendRequest;
45 import org.bitcoinj.wallet.Wallet;
46 import org.bitcoinj.wallet.listeners.WalletChangeEventListner;
47 import org.bitcoinj.wallet.listeners.WalletCoinsReceivedEventListener;
48 import org.spongeycastle.util.encoders.Hex;
49
50
51 public class OpReturnMain { // (BITCOINJ, 2017)
52
53     // set bitcoin wallet to either MainNet (main network where bitcoin have economic value)
54     // or TestNet (used for testing purposes and bitcoins do not have any value)
55     public static NetworkParameters params = TestNet3Params.get();
56     // public static NetworkParameters params = MainNetParams.get();
57

```



```

58 // set wallet name
59 public static final String APP_NAME = "Twinings";
60 private static final String TWININGS = APP_NAME.replaceAll("[^a-zA-Z0-9-]", "_") + "-" +
params.getPaymentProtocolId();
61
62 // declare new SPV (Simplified Payment Verification) bitcoinj app
63 public static WalletAppKit bitcoin;
64
65 public static void main(String[] args) {
66
67     //
68     setupWalletKit();
69
70     // start a runnable thread process that runs the wallet's event listeners.
71     // It is constantly listening for events that occur to bitcoin addresses that belong to the wallet
72     bitcoin.addListener(new Service.Listener() {
73         @Override
74         public void starting() {
75             super.starting();
76             System.out.println("starting");
77         }
78
79         @Override
80         public void running() {
81             super.running();
82             System.out.println("running: " + bitcoin.wallet().currentChangeAddress().toString());
83         }
84
85         @Override
86         public void stopping(Service.State from) {
87             super.stopping(from);
88             System.out.println("stopping");
89         }
90
91         @Override
92         public void terminated(Service.State from) {
93             super.terminated(from);
94             System.out.println("terminated");
95         }
96
97         @Override
98         public void failed(Service.State from, Throwable failure) {
99             super.failed(from, failure);
100             System.out.println("failed");
101         }
102     }, Runnable::run);
103     bitcoin.addListener(new Service.Listener() {
104     }, OpReturnRunnable::runLater);
105     bitcoin.startAsync();
106
107 }
108
109 public static void setupWalletKit() {
110
111     // create new SPV (Simplified Payment Verification) bitcoinj app or
112     // if seed wallet is non-null it means we are restoring from backup.
113     bitcoin = new WalletAppKit(params, new File("."), TWININGS) {
114         @Override
115         protected void onSetupCompleted() {
116             // Don't make the user wait for confirmations for now, as the intention is they're sending it
117             // their own money!
118             bitcoin.wallet().allowSpendingUnconfirmedTransactions();
119             System.out.println("WalletAppKit onSetupCompleted: " + bitcoin.wallet().currentChangeAddress().toString());
120             System.out.println("port: " + params.getPort());
121             System.out.println("wallet current balance: " + bitcoin.wallet().getBalance().toString());
122
123             // Java Persistence API instance for application database
124             EntityManagerFactory emf = Persistence.createEntityManagerFactory("apekato");
125             EntityManager em = emf.createEntityManager();
126
127             try {
128                 // register invalid data when wallet is initialized and when wallet is changed
129                 registerInvalidData(em);
130             } catch (Exception ex) {
131                 Logger.getLogger(OpReturnMain.class.getName()).log(Level.SEVERE, null, ex);
132             }
133         }
134     };
135 }

```

```

133     }
134     em.close(); emf.close();
135
136     // start wallet event listeners
137     walletListener();
138 }
139 };
140 // bitcoin.setBlockingStartup(false);
141 }
142
143 public static void registerInvalidData(EntityManager em) throws Exception{
144     List<OpReturn> opReturnList = GenericDaoJpa.findListByAttribute(
145         em,
146         OpReturn.class,
147         "status",
148         OpReturn.OpReturnStatus.OP_RETURN_STATUS_INVALID_DATA);
149     if (opReturnList.size() < OpReturn.OpReturnInvalidDataList.SIZE) {
150         int newInvalidDataNum = OpReturn.OpReturnInvalidDataList.SIZE - opReturnList.size();
151         if (newInvalidDataNum != 0){
152             for (int i = 0; i < newInvalidDataNum; i++){
153                 String freshReceiveAddress = bitcoin.wallet().freshReceiveAddress().toString();
154                 System.out.println("freshReceiveAddress: " + freshReceiveAddress);
155                 OpReturn opReturn = new OpReturn(
156                     "",
157                     freshReceiveAddress,
158                     OpReturn.OpReturnStatus.OP_RETURN_STATUS_INVALID_DATA,
159                     new Date());
160                 GenericDaoJpa.insert(em, opReturn);
161             }
162         }
163     }
164 }
165
166 public static void registerOpReturnData(EntityManager em) throws Exception{
167
168     List<OpReturn> opReturnList = GenericDaoJpa.findListByAttribute(
169         em,
170         OpReturn.class,
171         "status",
172         OpReturn.OpReturnStatus.OP_RETURN_STATUS_WAITING_TX);
173
174     for (OpReturn opReturn : opReturnList){
175         Set<Transaction> transactionSet = bitcoin.wallet().getTransactions(false);
176         Iterator<Transaction> iterator = transactionSet.iterator();
177         while(iterator.hasNext()) {
178             Transaction transaction = iterator.next();
179             List<TransactionOutput> transactionOutputs = transaction.getOutputs();
180             for (TransactionOutput to : transactionOutputs){
181                 Address addressFromP2PKHScript = to.getAddressFromP2PKHScript(params);
182                 if (addressFromP2PKHScript != null){
183                     if (addressFromP2PKHScript.toString().equals(opReturn.getAddress())){
184                         System.out.println("getAddressFromP2PKHScript: " + addressFromP2PKHScript.toString());
185                         timestampData(em, opReturn);
186                     }
187                 }
188
189                 // Address addressFromP2SH = to.getAddressFromP2SH(params);
190                 // if (addressFromP2SH != null){
191                 //     if (addressFromP2SH.toString().equals(opReturn.getAddress())){
192                 //         System.out.println("getAddressFromP2SH: " + addressFromP2SH.toString());
193                 //         timestampData(em, opReturn);
194                 //     }
195                 // }
196             }
197         }
198     }
199 }
200
201 public static void registerOpReturnData(EntityManager em, Transaction tx) throws Exception{
202
203     List<OpReturn> opReturnList = GenericDaoJpa.findListByAttribute(
204         em,
205         OpReturn.class,
206         "status",
207         OpReturn.OpReturnStatus.OP_RETURN_STATUS_WAITING_TX);
208

```

```

209     for (OpReturn opReturn : opReturnList){
210
211         for (TransactionOutput to : tx.getOutputs()){
212             Address addressFromP2PKHScript = to.getAddressFromP2PKHScript(params);
213             if (addressFromP2PKHScript != null){
214                 if (addressFromP2PKHScript.toString().equals(opReturn.getAddress())){
215                     System.out.println("getAddressFromP2PKHScript: " + addressFromP2PKHScript.toString());
216                     timestampData(em, opReturn);
217                 }
218             }
219         }
220     }
221 }
222
223
224 // timestamp data in the blockchain
225 public static void timestampData(EntityManager em, OpReturn opReturn) throws IOException,
InsufficientMoneyException, Exception {
226
227     // create a byte variable to convert a SHA 256 hash of a signed document into bytes
228     byte[] opReturnBytes = null;
229
230     // timestamp any text message in the blockchain
231     if (opReturn.getType().endsWith(OpReturn.OpReturnType.OP_RETURN_TYPE_TEXT)){
232         opReturnBytes = opReturn.getText().getBytes("UTF-8");
233     }
234     // timestamp an ICP-Brasil signed document in the blockchain
235     } else if (opReturn.getType().endsWith(OpReturn.OpReturnType.OP_RETURN_TYPE_NOTARIZATION)){
236         opReturnBytes = Hex.decode(opReturn.getText());
237     }
238
239     // Create a tx with an OP_RETURN output
240     Transaction tx = new Transaction(params);
241     tx.addOutput(Coin.ZERO, ScriptBuilder.createOpReturnScript(opReturnBytes));
242
243     System.out.println("wallet before tx: " + bitcoin.wallet().getBalance().toString());
244
245     // send wallet information regarding timestamping data.
246     SendRequest req = SendRequest.forTx(tx);
247
248     // set timestamp trasaction fee (0.00015 BTC)
249     BigDecimal sendfee = opReturn.getFee().setScale(5, RoundingMode.HALF_EVEN);
250
251     // send timestamp transaction
252     req.feePerKb = Coin.parseCoin(sendfee.toString());
253
254 //     Coin c = req.feePerKb;
255 //     if (c.value < 15000) {
256 //         long add = 15000 - c.value;
257 //         req.feePerKb.add(Coin.valueOf(add));
258 //     } else if (c.value > 15000) {
259 //         long subtract = c.value - 15000;
260 //         req.feePerKb.add(Coin.valueOf(subtract));
261 //     }
262 //     Coin c2 = req.feePerKb;
263 // Send it to the Bitcoin network
264
265     // get result information of timestamp transaction
266     Wallet.SendResult sendResult = bitcoin.wallet().sendCoins(req);
267
268 //     long fee = sendResult.tx.getFee().longValue();
269
270     System.out.println("getHashAsString: " + sendResult.tx.getHashAsString());
271
272     // checks if timestamp transaction was successful
273     if (sendResult.tx.getHashAsString() != null){
274
275         // set timestamp opReturn object for persistence in application database
276         // change timestamp opReturn object status to REGISTERED
277         opReturn.setStatus(OpReturn.OpReturnStatus.OP_RETURN_STATUS_REGISTERED);
278         // set timestamp transaction ID into timestamp opReturn object
279         opReturn.setTxId(sendResult.tx.getHashAsString());
280         // persist timestamp opReturn object in application database
281         String ok = GenericDaoJpa.updateWithoutTx(em, OpReturn.class, opReturn);
282
283 //         TxOpReturn txOpReturn = new TxOpReturn(

```

```

284 //      opReturn.getText(),
285 //      opReturn.getAddress(),
286 //      opReturn.getStatus(),
287 //      opReturn.getType(),
288 //      opReturn.getEmail(),
289 //      new Date(),
290 //      sendResult.tx.getHashAsString(),
291 //      opReturn.getFee(),
292 //      opReturn.getLoginId());
293 //      GenericDaoJpa.insert(em, txOpReturn);
294
295 // send email to user with information regarding timestamp transation
296 if (opReturn.getType().equals(OpReturn.Type.OP_RETURN_TYPE_NOTARIZATION)){
297     File file = new File(ProjService.RBPATH);
298     URL[] urls = {file.toURI().toURL()};
299     ClassLoader loader = new URLClassLoader(urls);
300
301     ResourceBundle rb;
302     String language = opReturn.getLang();
303     if (language.equals("pt")){
304         rb = ResourceBundle.getBundle("SystemMessages", Locale.forLanguageTag("pt"), loader);
305     } else {
306         rb = ResourceBundle.getBundle("SystemMessages", Locale.forLanguageTag("en"), loader);
307     }
308     String emailSubject = rb.getString("email_body_op_return_subject_notarize");
309     StringBuilder emailBodyOpReturn = new StringBuilder();
310     emailBodyOpReturn.append(rb.getString("email_body_hi"));
311     emailBodyOpReturn.append(",");
312     emailBodyOpReturn.append("<br><br><br><br>");
313     emailBodyOpReturn.append(rb.getString("email_body_op_return_subject_notarize"));
314     emailBodyOpReturn.append(": ");
315     emailBodyOpReturn.append("<br><br><br><br>");
316     emailBodyOpReturn.append(rb.getString("email_body_op_return_transaction_id"));
317     emailBodyOpReturn.append(" ");
318     emailBodyOpReturn.append(opReturn.getTxId());
319     emailBodyOpReturn.append("<br><br><br><br>");
320     emailBodyOpReturn.append(rb.getString("email_body_op_return_search"));
321     emailBodyOpReturn.append(" ");
322
323     if (ProjService.ADDRESS == ProjService.AddressType.MAIN){
324         emailBodyOpReturn.append("https://chain.so/tx/BTC/");
325     }else if (ProjService.ADDRESS == ProjService.AddressType.TESTNET){
326         emailBodyOpReturn.append("https://chain.so/tx/BTCTEST/");
327     }
328     emailBodyOpReturn.append(opReturn.getTxId());
329
330     emailBodyOpReturn.append("<br><br><br><br>");
331     emailBodyOpReturn.append(rb.getString("email_body_end"));
332
333     JavaMailThread javaMailThread_1 = new JavaMailThread(opReturn.getEmail(), emailSubject,
emailBodyOpReturn.toString());
334     ExecutorService threadExecutor = Executors.newCachedThreadPool();
335     threadExecutor.execute(javaMailThread_1);
336     threadExecutor.shutdown();
337 }else if (opReturn.getType().equals(OpReturn.Type.OP_RETURN_TYPE_TEXT)){
338     File file = new File(ProjService.RBPATH);
339     URL[] urls = {file.toURI().toURL()};
340     ClassLoader loader = new URLClassLoader(urls);
341
342     ResourceBundle rb;
343     String language = opReturn.getLang();
344     if (language.equals("pt")){
345         rb = ResourceBundle.getBundle("SystemMessages", Locale.forLanguageTag("pt"), loader);
346     } else {
347         rb = ResourceBundle.getBundle("SystemMessages", Locale.forLanguageTag("en"), loader);
348     }
349     String emailSubject = rb.getString("email_body_op_return_subject_message");
350     StringBuilder emailBodyOpReturn = new StringBuilder();
351     emailBodyOpReturn.append(rb.getString("email_body_hi"));
352     emailBodyOpReturn.append(",");
353     emailBodyOpReturn.append("<br><br><br><br>");
354     emailBodyOpReturn.append(rb.getString("email_body_op_return_subject_message"));
355     emailBodyOpReturn.append(": ");
356     emailBodyOpReturn.append("<br><br><br><br>");
357     emailBodyOpReturn.append(rb.getString("email_body_op_return_transaction_id"));
358     emailBodyOpReturn.append(" ");

```

```

359     emailBodyOpReturn.append(opReturn.getTxId());
360     emailBodyOpReturn.append("<br><br><br><br>");
361     emailBodyOpReturn.append(rb.getString("email_body_op_return_search"));
362     emailBodyOpReturn.append(" ");
363
364     if (ProjService.ADDRESS == ProjService.AddressType.MAIN){
365         emailBodyOpReturn.append("https://chain.so/tx/BTC/");
366     }else if (ProjService.ADDRESS == ProjService.AddressType.TESTNET){
367         emailBodyOpReturn.append("https://chain.so/tx/BTCTEST/");
368     }
369     emailBodyOpReturn.append(opReturn.getTxId());
370
371     emailBodyOpReturn.append("<br><br><br><br>");
372     emailBodyOpReturn.append(rb.getString("email_body_end"));
373
374     JavaMailThread javaMailThread_1 = new JavaMailThread(opReturn.getEmail(), emailSubject,
emailBodyOpReturn.toString());
375     ExecutorService threadExecutor = Executors.newCachedThreadPool();
376     threadExecutor.execute(javaMailThread_1);
377     threadExecutor.shutdown();
378 }
379
380 }
381
382 System.out.println("wallet after tx: " + bitcoin.wallet().getBalance().toString());
383
384 }
385
386 public static void walletListener() {
387 // bitcoin.wallet().addChangeEventEventListener(new WalletChangeEventEventListener() {
388 // @Override
389 // public void onWalletChanged(Wallet wallet) {
390 // System.out.println("onWalletChanged");
391 //
392 // EntityManagerFactory emf = Persistence.createEntityManagerFactory("apekato");
393 // EntityManager em = emf.createEntityManager();
394 //
395 // try {
396 // registerOpReturnData(em);
397 // // register invalid data when wallet is initialized and when wallet is changed
398 // registerInvalidData(em);
399 // } catch (Exception ex) {
400 // Logger.getLogger(OpReturnMain.class.getName()).log(Level.SEVERE, null, ex);
401 // }
402 //
403 // em.close();
404 // emf.close();
405 // }
406 // });
407
408 // shows an event listener
409 // that is triggered when a wallet address receives bitcoins from a user
410 // who requests a signed document authentication service
411 bitcoin.wallet().addCoinsReceivedEventListener(new WalletCoinsReceivedEventListener() {
412 @Override
413
414
415 public void onCoinsReceived(Wallet wallet, Transaction tx, Coin prevBalance, Coin newBalance) {
416 System.out.println("onCoinsReceived");
417
418 EntityManagerFactory emf = Persistence.createEntityManagerFactory("apekato");
419 EntityManager em = emf.createEntityManager();
420
421 try {
422
423 // retrieve in application database a wallet address that has received a
424 // bitcoin payment and timestamps a signed document's SHA 256 Hash
425 // through the OP_RETURN script operation code and updates the
426 // status of a signed document's timestamp process to "REGISTERED" database
427 registerOpReturnData(em, tx);
428
429 // register invalid data (i.e. new wallet addresses)
430 // when wallet is initialized or when a wallet status changes to "REGISTERED"
431 registerInvalidData(em);
432 } catch (Exception ex) {
433 Logger.getLogger(OpReturnMain.class.getName()).log(Level.SEVERE, null, ex);

```

```

434     }
435
436     em.close();
437     emf.close();    }
438     });
439 }
440 }
441

```

APPENDIX 4 – VERIFYNOTARIZATIONASYNC TASK OF ANDROID APP

```

1  package a.apkt.asynctask;
2
3  import android.animation.Animator;
4  import android.animation.AnimatorListenerAdapter;
5  import android.app.Activity;
6  import android.content.Intent;
7  import android.os.AsyncTask;
8  import android.os.Bundle;
9  import android.view.Gravity;
10 import android.view.MenuItem;
11 import android.view.View;
12 import android.widget.LinearLayout;
13 import android.widget.ScrollView;
14 import android.widget.Toast;
15
16 import com.google.gson.Gson;
17
18 import java.io.FileNotFoundException;
19 import java.io.IOException;
20 import java.io.InputStream;
21 import java.io.InputStreamReader;
22 import java.net.URL;
23
24 import javax.net.ssl.HttpURLConnection;
25
26 import a.apkt.ItemFragment;
27 import a.apkt.MainActivity;
28 import a.apkt.OpReturnEditActivity;
29 import a.apkt.R;
30 import a.apkt.VerifyNotarizationActivity;
31 import a.apkt.model.Tx;
32 import a.apkt.model.OpReturn;
33 import a.apkt.service.ProjService;
34 import a.apkt.service.StaticVars;
35
36 public class VerifyNotarizationAsyncTask extends AsyncTask<String, Void, Boolean> {
37
38     private Activity activity;
39     private View mProgressView;
40     private ScrollView scrollView;
41     private LinearLayout linearLayout;
42     private MenuItem actionAddItem;
43     private Tx tx;
44     private String txId;
45     private boolean iOExceptionWrongFormat = false;
46     private boolean fileNotFoundException = false;
47     private ItemFragment itemFragment;
48     private OpReturn opReturn;
49
50     public VerifyNotarizationAsyncTask(Activity activity, MenuItem actionAddItem, String txId){
51         this.activity = activity;
52         this.actionAddItem = actionAddItem;
53         this.txId = txId;
54     }
55
56     public VerifyNotarizationAsyncTask(Activity activity, ItemFragment itemFragment, OpReturn opReturn){
57         this.activity = activity;
58         this.itemFragment = itemFragment;
59         this.opReturn = opReturn;

```

```

60     txId = opReturn.getTxId();
61 }
62
63
64 protected void onPreExecute() {
65     if (activity.getClass() == MainActivity.class){
66         mProgressBar = activity.findViewById(R.id.activity_progress);
67         if (itemFragment != null){
68             linearLayout = (LinearLayout) activity.findViewById(R.id.linear_layout);
69         } else {
70             scrollView = (ScrollView) activity.findViewById(R.id.scroll_view);
71         }
72         showProgress(true);
73     }
74 }
75 }
76
77 @Override
78 protected Boolean doInBackground(String... params) {
79
80     try {
81
82         // url to retrieve a transaction id of a document timestamped
83         // on bitcoin's blockchain using BlockCypher's API
84         String url_ = ProjService.BLOCK_CYPHER_ENDPOINT + "txs/" + txId + "?token=" +
ProjService.BLOCK_CYPHER_TOKEN;
85
86         // get a pointer to a "resource" on the World Wide Web
87         URL url = new URL(url_);
88
89         // send request to BlockCypher's API
90         HttpURLConnection urlConnection = (HttpURLConnection)url.openConnection();
91
92         // load into InputStream data of timestamp transaction on bitcoin's blockchain
93         InputStream in = urlConnection.getInputStream();
94
95         // convert InputStream to InputStreamReader
96         InputStreamReader reader =
97             new InputStreamReader(in);
98
99         // load Tx (transaction) object that has detailed information of transaction id of
100        // a document timestamped on bitcoin's blockchain using BlockCypher's API
101        tx = new Gson().fromJson(reader, Tx.class);
102
103    } catch (FileNotFoundException exception) {
104        fileNotFoundException = true;
105        return false;
106    } catch (IOException exception) {
107        iOExceptionWrongFormat = true;
108        return false;
109    } catch (Exception exception) {
110        exception.printStackTrace(); // show exception details
111        return false;
112    }
113
114    return true;
115 }
116
117 @Override
118 protected void onPostExecute(final Boolean success) {
119
120
121     if (success) {
122
123         if (activity.getClass() == MainActivity.class && itemFragment == null){
124             showProgress(false);
125             MainActivity mainActivity = (MainActivity) activity;
126             mainActivity.setVerifyNotarizationTx(tx);
127             Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT); // (ETERNITY WALL, 2016)
128             intent.setType("*/*"); // (ETERNITY WALL, 2016)
129             activity.startActivityForResult(intent,
StaticVars.OPEN_DOCUMENT_VERIFY_NOTARIZATION_ACTIVITY_RESULT);
130         } else if (activity.getClass() == MainActivity.class && itemFragment != null) {
131             showProgress(false);
132             String txBlockCypherJson = new Gson().toJson(tx);

```



```

133     String opReturnJson = new Gson().toJson(opReturn);
134     Intent intent = new Intent(activity, OpReturnEditActivity.class);
135     Bundle args = new Bundle();
136     args.putString("opReturnJson", opReturnJson);
137     args.putString("txBlockCypherJson", txBlockCypherJson);
138     // args.putString("loginStringJson", loginStringJson);
139     intent.putExtras(args);
140     activity.startActivity(intent);
141     } else if (activity.getClass() == VerifyNotarizationActivity.class) { // doesn't use showProgress() because it uses
SwipeRefreshLayout
142         VerifyNotarizationActivity verifyNotarizationActivity = (VerifyNotarizationActivity) activity;
143         verifyNotarizationActivity.setTx(tx);
144     } else if (activity.getClass() == OpReturnEditActivity.class) { // doesn't use showProgress() because it uses
SwipeRefreshLayout
145         OpReturnEditActivity OpReturnEditActivity = (OpReturnEditActivity) activity;
146         OpReturnEditActivity.setTx(tx);
147     }
148 }
149 else if (!success) {
150     showProgress(false);
151     if (fileNotFoundException) {
152         Toast toast = Toast.makeText(activity, activity.getString(R.string.toast_tx_id_invalid), Toast.LENGTH_LONG);
153         toast.setGravity(Gravity.TOP, 0, 0);
154         toast.show();
155     } else if (IOExceptionWrongFormat){
156         Toast toast = Toast.makeText(activity, activity.getString(R.string.toast_tx_id_wrong_format),
Toast.LENGTH_LONG);
157         toast.setGravity(Gravity.TOP, 0, 0);
158         toast.show();
159     }
160 }
161 }
162
163 public void showProgress(final boolean show) {
164     int shortAnimTime = activity.getResources().getInteger(android.R.integer.config_shortAnimTime);
165     mProgressBar.setVisibility(show ? View.VISIBLE : View.GONE);
166     if (itemFragment != null){
167         linearLayout.setVisibility(show ? View.GONE : View.VISIBLE);
168     }else {
169         scrollView.setVisibility(show ? View.GONE : View.VISIBLE); // scrollView from VerifyNotarizationFragment
170         actionAddItem.setEnabled(show ? false : true); // actionAddItem from VerifyNotarizationFragment
171     }
172     mProgressBar.animate().setDuration(shortAnimTime).alpha(
173         show ? 1 : 0).setListener(new AnimatorListenerAdapter() {
174         @Override
175         public void onAnimationEnd(Animator animation) {
176             mProgressBar.setVisibility(show ? View.VISIBLE : View.GONE);
177         }
178     });
179 }
180 }
181

```