Universitat
de les Illes Balears

DOCTORAL THESIS

2024

BLOCKCHAIN-BASED E-COMMERCE PROTOCOLS

Miquel Àngel Cabot Nadal

Universitat
de les Illes Balears

DOCTORAL THESIS

2024

Doctoral Programme in Information and Communications Technology

BLOCKCHAIN-BASED E-COMMERCE PROTOCOLS

Miquel Àngel Cabot Nadal

Thesis supervisor: Macià Mut Puigserver
Thesis supervisor: Maria Magdalena Payeras Capellà
Thesis tutor: Macià Mut Puigserver

Doctor by the Universitat de les Illes Balears

Dr **Macià Mut Puigserver**, of Universitat de les Illes Balears

I DECLARE:

That the *Blockchain-based e-commerce protocols* thesis, presented by Miquel Àngel Cabot Nadal to obtain a doctoral degree, has been completed under my supervision and meets the requirements to opt for a doctorate.

For all intents and purposes, I hereby sign this document.

Signature

July 16th, 2024

Dra **Maria Magdalena Payeras Capellà**, of Universitat de les Illes Balears

I DECLARE:

That the *Blockchain-based e-commerce protocols* thesis, presented by Miquel Àngel Cabot Nadal to obtain a doctoral degree, has been completed under my supervision and meets the requirements to opt for a doctorate.

For all intents and purposes, I hereby sign this document.

Signature

July 16th, 2024

Dedicated to my family

ACKNOWLEDGMENTS

I express my sincere gratitude to Professors Macià Mut and Maria Magdalena Payeras for providing me with this invaluable opportunity and for their guidance throughout the development of this thesis. Without their indispensable help, patience, and steadfast support, I would not have successfully navigated and completed this academic journey.

I extend my appreciation to those who collaborated on the papers associated with this thesis: Llorenç Huguet, Rosa Pericàs, and Jaume Ramis. Their generous collaboration and valuable time significantly enhanced the depth of knowledge within this thesis.

I am eternally grateful to my parents, Antoni and Catalina, who have set an inspiring example of perseverance and constant determination. I owe them a tremendous debt of gratitude for the invaluable lessons they have imparted to me.

A heartfelt and final expression of gratitude is extended to my partner, Maria, whose unwavering support sustained me through the highs and lows of this journey. Additionally, my deepest appreciation goes to my children, Joan and Aina, who continually inspire me to look ahead and strive for self-improvement each day.

Each individual mentioned has played a crucial role in this academic endeavor, and for that, I am profoundly thankful.

Moltíssimes gràcies a tots!!

CONTENTS

ABSTRACT

Blockchain is a decentralized distributed database where information, stored in blocks, is cryptographically linked to ensure the security of transactions. This technology represents an evolution in many areas of technology, including digital services, the economy, electronic commerce, and other fields. The decentralized nature of blockchain ensures that no single entity has control over the entire network, thereby enhancing security and ensuring robustness. On the other hand, the rapid growth of e-commerce has led to increasing demand for secure and efficient online transactions. The emergence of blockchain technology has provided a promising solution to the challenges of online transactions, including security, transparency, auditability, and immutability. This thesis explores the potential of blockchain technology in the e-commerce industry and proposes a set of blockchain-based e-commerce protocols that can enhance the security and efficiency of online transactions.

This thesis first reviews the existing literature on blockchain technology and its applications in various domains. It analyses a range of blockchain-based or blockchain-related technologies and standards, that will help us to create a novel set of protocols that need decentralization, as well as robust security and privacy mechanisms, to enhance fair exchange operations. The technologies and standards that will be used to create these protocols can vary from token standards such as Non-Fungible tokens, Soulbound tokens, and Rejectable tokens, design patterns such as *Factory smart contracts*, to other cryptographic protocols or standards like hash functions, Merkle trees, Elliptic Curve Cryptography, and Identity-Based Encryption.

As a shared, immutable database, blockchain can be utilized for various purposes, including the elimination of the need for a Trusted Third Party (TTP), among other use cases that will be enumerated in this study. Some new protocols and standards are proposed, like certified notifications or registered eDeliveries, contract signing, micropayments, and confidential digital identity management, all of them related to e-commerce. All of these are possible thanks to more advanced blockchains like Ethereum, which is a globally decentralized computing infrastructure that executes programs called smart contracts. Ethereum, as a Turing-complete machine, can act as a general-purpose computer and gives us a lot of new opportunities in the information and computing decentralization objective.

But blockchain technology also has some problems and challenges to be solved, like scalability and gas consumption. These are critical issues that can impact the efficiency and usability of blockchain protocols. For this reason, this study will also analyze the proposed protocols from the performance point of view. A thorough analysis of security properties will also be carried out, such as integrity, authenticity, non-repudiation, and fairness.

Taking all of this into consideration, this thesis will contribute to the understanding of the potential of blockchain technology in the e-commerce industry and will provide practical solutions for enhancing the security and efficiency of online transactions. By addressing these critical areas, this research endeavors to pave the way for a more decentralized, transparent, trustworthy, and robust e-commerce ecosystem, fostering innovation and driving the growth of online businesses in the digital age.

RESUM

Blockchain és una base de dades distribuïda descentralitzada on la informació, emmagatzemada en blocs, s'enllaça criptogràficament per garantir la seguretat de les transaccions. Aquesta tecnologia representa una evolució en moltes àrees de la tecnologia, inclosos els serveis digitals, l'economia, el comerç electrònic, i altres camps. La naturalesa descentralitzada de la cadena de blocs garanteix que cap entitat tingui control sobre tota la xarxa, millorant així la seguretat i garantint la robustesa. D'altra banda, el ràpid creixement del comerç electrònic ha fet augmentar la demanda de transaccions en línia segures i eficients. L'aparició de la tecnologia blockchain ha proporcionat una solució prometedora als reptes de les transaccions en línia, incloent seguretat, transparència, auditabilitat i immutabilitat. Aquesta tesi explora el potencial de la tecnologia blockchain a la indústria del comerç electrònic i proposa un conjunt de protocols de comerç electrònic basats en blockchain que poden millorar la seguretat i l'eficiència de les transaccions en línia.

Aquesta tesi revisa primer la literatura existent sobre tecnologia blockchain i les seves aplicacions en diversos dominis. Analitza una sèrie de tecnologies i estàndards basats o relacionats amb blockchain, que ens ajudaran a crear un nou conjunt de protocols que necessiten descentralització, així com mecanismes de seguretat i privadesa robusts, per millorar les operacions d'intercanvi just. Les tecnologies i estàndards que s'utilitzaran per crear aquests protocols poden variar des d'estàndards de *tokens*, com ara *tokens* no fungibles, *Soulbound tokens* i *tokens* rebutjables, patrons de disseny com *Factory smart contracts*, fins a altres protocols criptogràfics o estàndards com funcions hash, arbres Merkle, criptografia de corba el·líptica i xifratge basat en la identitat.

Com a base de dades compartida i immutable, la cadena de blocs es pot utilitzar per a diversos propòsits, inclosa l'eliminació de la necessitat d'una Tercera Part Confiable o *Trusted Third Party (TTP)*, entre altres casos d'ús que s'enumeraran en aquest estudi. Es proposen alguns nous protocols i estàndards, com ara notificacions certificades o lliuraments electrònics registrats (*registered eDeliveries*), signatura de contractes, micropagaments, i gestió d'identitat digital confidencial, tots ells relacionats amb el comerç electrònic. Tot això és possible gràcies a cadenes de blocs més avançades com Ethereum, que és una infraestructura informàtica descentralitzada a nivell mundial que executa programes anomenats *smart contracts*. Ethereum, com a màquina completa de Turing, pot actuar com un ordinador de propòsit general i ens ofereix moltes oportunitats noves en l'objectiu de descentralització de la informació i la computació.

Però la tecnologia blockchain també té alguns problemes i reptes per resoldre, com l'escalabilitat i el consum de gas. Aquests són problemes crítics que poden afectar l'eficiència i la usabilitat dels protocols blockchain. Per aquest motiu, aquest estudi també analitzarà els protocols proposats des del punt de vista del rendiment. També es

realitzarà una anàlisi exhaustiva de les propietats de seguretat, com ara la integritat, l'autenticitat, el no repudi i l'equitat.

Tenint en compte tot això, aquesta tesi contribuirà a entendre el potencial de la tecnologia blockchain en la indústria del comerç electrònic i proporcionarà solucions pràctiques per millorar la seguretat i l'eficiència de les transaccions en línia. En abordar aquestes àrees crítiques, aquesta investigació intenta obrir el camí cap a un ecosistema de comerç electrònic més descentralitzat, transparent, fiable i robust, fomentant la innovació i impulsant el creixement de les empreses en línia a l'era digital.

RESUMEN

Blockchain es una base de datos distribuida descentralizada donde la información, almacenada en bloques, se enlaza criptográficamente para garantizar la seguridad de las transacciones. Esta tecnología representa una evolución en muchas áreas de la tecnología, incluidos los servicios digitales, la economía, el comercio electrónico, y otros campos. La naturaleza descentralizada de la cadena de bloques garantiza que ninguna entidad tenga control sobre toda la red, mejorando así la seguridad y garantizando la robustez. Por otra parte, el rápido crecimiento del comercio electrónico ha hecho aumentar la demanda de transacciones online seguras y eficientes. La aparición de la tecnología blockchain ha proporcionado una solución prometedora a los retos de las transacciones online, incluyendo seguridad, transparencia, auditabilidad e inmutabilidad. Esta tesis explora el potencial de la tecnología blockchain en la industria del comercio electrónico y propone un conjunto de protocolos de comercio electrónico basados en blockchain que pueden mejorar la seguridad y la eficiencia de las transacciones online.

Esta tesis revisa primero la literatura existente sobre tecnología blockchain y sus aplicaciones en varios dominios. Analiza una serie de tecnologías y estándares basados o relacionados con blockchain, que nos ayudarán a crear un nuevo conjunto de protocolos que necesitan descentralización, así como mecanismos de seguridad y privacidad robustos, para mejorar las operaciones de intercambio justo. Las tecnologías y estándares que se utilizarán para crear estos protocolos pueden variar desde estándares de *tokens*, tales como *tokens* no fungibles, *Soulbound tokens* y *tokens* rechazables, patrones de diseño como *Factory smart contracts*, hasta otros protocolos criptográficos o estándares como funciones hash, árboles Merkle, criptografía de curva elíptica y cifrado basado en la identidad.

Como base de datos compartida e inmutable, la cadena de bloques se puede utilizar para diversos fines, incluida la eliminación de la necesidad de una Tercera Parte Confiable o *Trusted Third Party (TTP)*, entre otros casos de uso que se enumerarán en este estudio. Se proponen algunos nuevos protocolos y estándares como notificaciones certificadas o entregas electrónicas registradas (*registered eDeliveries*), firma de contratos, micropagos, y gestión de identidad digital confidencial, todos ellos relacionados con el comercio electrónico. Todo esto es posible gracias a cadenas de bloques más avanzadas como Ethereum, que es una infraestructura informática descentralizada a nivel mundial que ejecuta programas llamados *smart contracts*. Ethereum, como máquina completa de Turing, puede actuar como un ordenador de propósito general y nos ofrece muchas nuevas oportunidades en el objetivo de descentralización de la información y la computación.

Pero la tecnología blockchain también tiene algunos problemas y retos por re-

solver, como la escalabilidad y el consumo de gas. Estos son problemas críticos que pueden afectar a la eficiencia y la usabilidad de los protocolos blockchain. Por este motivo, este estudio analizará también los protocolos propuestos desde el punto de vista del rendimiento. También se realizará un análisis exhaustivo de las propiedades de seguridad, tales como la integridad, la autenticidad, el no repudio y la equidad.

Teniendo en cuenta todo esto, esta tesis contribuirá a entender el potencial de la tecnología blockchain en la industria del comercio electrónico y proporcionará soluciones prácticas para mejorar la seguridad y la eficiencia de las transacciones online. Al abordar estas áreas críticas, esta investigación intenta abrir el camino hacia un ecosistema de comercio electrónico más descentralizado, transparente, fiable y robusto, fomentando la innovación e impulsando el crecimiento de las empresas online en la era digital.

PUBLICATION LIST

The list of publications and other results generated by this thesis is presented below.

Published journal articles

1. M. M. Payeras-Capellà, M. Mut-Puigserver, and M. A. Cabot-Nadal, "Blockchain-Based System for Multiparty Electronic Registered Delivery Services," *IEEE Access*, vol. 7, pp. 95825-95843, 2019. doi: 10.1109/ACCESS.2019.2929101

2. M. Mut-Puigserver, M. A. Cabot-Nadal, and M. M. Payeras-Capellà, "Removing the Trusted Third Party in a Confidential Multiparty Registered eDelivery Protocol Using Blockchain," *IEEE Access*, vol. 8, pp. 106855-106871, 2020. doi: 10.1109/ACCESS.2020.3000558

3. M. M. Payeras-Capellà, M. Mut-Puigserver, M. A. Cabot-Nadal, and L. Huguet Rotger, "Blockchain-Based Confidential Multiparty Contract Signing Protocol Without TTP Using Elliptic Curve Cryptography," *The Computer Journal*, vol. 65, no. 10, pp. 2755-2768, 2022. doi: 10.1093/comjnl/bxab112

4. M. A. Cabot-Nadal, M. Mut-Puigserver, M. M. Payeras-Capellà, and R. Pericàs-Gornals, "Confidential Certified Notification Protocol using Rejectable Soulbound Tokens and Identity-based Cryptography," *IEEE Access*, vol. 11, pp. 142495-142514, 2023. doi: 10.1109/ACCESS.2023.3342446

Submitted journal articles

1. M. A. Cabot-Nadal, M. M. Payeras-Capellà, M. Mut-Puiserver, R. Pericàs-Gornals, J. Ramis-Bibiloni, B. Playford, S. Gerske, "Safeguarding Privacy with Soulbound Tokens and Zero-Knowledge Proofs: Fully Developed Identity-Related Attribute Verification Protocol," Submitted to *Blockchain: Research and Applications* in March, 2024

Published conference proceedings

1. M. Mut-Puigserver, M. M. Payeras-Capellà, and M. A. Cabot-Nadal, "Blockchain-Based Fair Certified Notifications," *Data Privacy Management, Cryptocurrencies and Blockchain Technology (Lecture Notes in Computer Science), Springer, Barcelona*, vol. 11025, pp. 20–37, 2018. doi: 10.1007/978-3-030-00305-0_2

2. M. Mut Puigserver, M. M. Payeras-Capellà, and M. A. Cabot-Nadal, "Notificaciones Certificadas sobre Blockchain," *Actas XV Reunión Española de Criptografia*

y Seguridad de la Información (RECSI), Granada, pp. 251-256, 2018. ISBN: 978-84-09-02463-6

3. M. M. Payeras-Capellà, M. Mut-Puigserver, and M. A. Cabot-Nadal, "Smart Contract for Multiparty Fair Certified Notifications," *2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW), Takayama, Japan*, pp. 459-465, 2018. doi: 10.1109/CANDARW.2018.00089

4. M. M. Payeras-Capellà, M. Mut-Puigserver, and M. A. Cabot-Nadal, "Protocolos de Comercio Electronico Basados en Blockchain," *Actas de las XIV Jornadas de Ingenieria Telemàtica (JITEL), Zaragoza*, 2019. Available: `jitel2019.i3a.es`

5. M. Mut-Puigserver, M. M. Payeras-Capellà, and M. A. Cabot-Nadal, "Blockchain-Based Contract Signing Protocol for Confidential Contracts," *IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA), Abu Dhabi, United Arab Emirates*, pp. 1-6, 2019. doi: 10.1109/AICCSA47632.2019.9035363

6. M. Mut-Puigserver, M. A. Cabot-Nadal, M. M. Payeras-Capellà, and L. Huguet Rotger, "Implementación de un Protocolo Confidencial y Multiparte para Entregas Certificadas sobre la Blockchain Ethereum," *Actas de la XVI Reunión Española sobre Criptología y Seguridad de la Información (RECSI), Lleida*, pp. 125-131, 2021. ISBN: 978-84-09-29150-2

7. M. M. Payeras-Capellà, M. A. Cabot-Nadal, and M. Mut-Puigserver, "Protocolo Basado en Blockchain para la Gestión de Canales para Microcompras Equitativas," *Actas de las XV Jornadas de Ingeniería Telemática (JITEL), A Coruña*, pp. 184-191, 2021. ISBN: 978-84-09-35131-2

8. M. Mut-Puigserver, R. Pericàs-Gornals, M. M. Payeras-Capellà, and M. A. Cabot-Nadal, "Improving the Efficiency of a Blockchain-Based Confidential Registered e-Delivery Protocol," *Advanced Information Systems Engineering Workshops (CAiSE 2022), Leuven, Belgium. Lecture Notes in Business Information Processing, Springer, Cham*, vol. 451, 2022. doi: 10.1007/978-3-031-07478-3_1

9. M. A. Cabot-Nadal, M. M. Payeras-Capellà, M. Mut-Puigserver, and A. Soto-Fernández, "Improving the Token ERC-721 Implementation for Selective Receipt: Rejectable NFTs," *6th International Conference on System Reliability and Safety (ICSRS), Venice, Italy*, 2022. doi: 10.1109/ICSRS56243.2022.10067494

10. M. A. Cabot-Nadal, B. Playford, M. M. Payeras-Capellà, S. Gerske, M. Mut-Puigserver, and R. Pericàs-Gornals, "Private Identity-Related Attribute Verification Protocol Using SoulBound Tokens and Zero-Knowledge Proofs," *7th Cyber Security in Networking Conference (CSNet), Montréal, QC, Canada*, pp. 153-156, 2023. doi: 10.1109/CSNet59123.2023.10339754

Patents

- M. M. Payeras-Capellà, M. Mut-Puigserver, L. Huguet Rotger, and M. A. Cabot-Nadal, "Method for notifications and certified deliveries based on blockchain technology". Patent number: ES2802420. Issue date: 25/04/2022. Available: `consultas2.oepm.es`

Research projects

- Project TIN2014-54945-R "Safe Access to Tourist Services (Access Tur)", funded by Ministerio de Economía y Competitividad (MINECO)
- Project RTI2018-097763-B-I00 "Fair Exchange, Loyalty and TIckets with blockCHAIN (FeltiCHAIN)", funded by Ministerio de Ciencia e Innovación (MCIN), Agencia Estatal de Investigacion (AEI) and European Regional Development Funds (ERDF)
- Project PID2021-122394OB-I00 "Security services based on blockchain technology (BLOBSEC)", funded by Ministerio de Ciencia e Innovación (MCIN), Agencia Estatal de Investigacion (AEI) and European Regional Development Funds (ERDF)

Research awards

- Best research paper of the XV Spanish Meeting on Cryptology and Information Security (RECSI). M. Mut Puigserver, M. M. Payeras-Capellà, and M. A. Cabot-Nadal, "Notificaciones Certificadas sobre Blockchain," *Actas XV Reunión Española de Criptografia y Seguridad de la Información (RECSI), Granada*, pp. 251-256, 2018. Award information: https://diari.uib.cat/Hemeroteca/Premi-per-a-un-estudi-sobre-laplicacio-de-cadenes.cid564082
- IEEE Blockchain Spain Initiative Award for Best Research Article 2020. M. M. Payeras-Capellà, M. Mut-Puigserver, and M. A. Cabot-Nadal, "Blockchain-Based System for Multiparty Electronic Registered Delivery Services," *IEEE Access*, vol. 7, pp. 95825-95843, 2019. Award information: https://ieeespain.org/ganadores-premios-ieee-blockchain-spain-2020/

Acronyms

AI Artificial Intelligence

API Application Programming Interface

ABI Application Binary Interface

B2B Business-to-Business

B2C Business-to-Consumer

B2G Business-to-Government

BSC Binance Smart Chain

BNB Binance Coin

BTC Bitcoin

C2B Consumer-to-Business

C2C Consumer-to-Consumer

CEX Centralized cryptocurrency Exchange

CID Content Identifier

DAM Distributed Authentication Mechanism

DAO Decentralized Autonomous Organization

DeFi Decentralized Finance

DEX Decentralized cryptocurrency Exchange

DID Decentralized Identifier

DLIES Discrete Logarithm Integrated Encryption Scheme

DLT Distributed Ledger Technology

DNS Domain Name System

DoS Denial of Service

DDoS Distributed Denial of Service

ECC Elliptic Curve Cryptography

ECDLP Elliptic Curve Discrete Logarithm Problem

ECDH Elliptic Curve Diffie-Hellman

ECDSA Elliptic Curve Digital Signature Algorithm

ECIES Elliptic Curve Integrated Encryption Scheme

EDI Electronic Data Interchange

EIP Ethereum Improvement Proposal

ENS Ethereum Name Service

EOA Externally-Owned Account

ERC Ethereum Request for Comment

ETH Ether

EVM Ethereum Virtual Machine

HEX Hybrid cryptocurrency Exchange

HTLC Hashed Timelock Contract

HTTP Hypertext Transfer Protocol

IBE Identity-Based Encryption

IDE Integrated Development Environment

IMS Identity Management System

IoT Internet of Things

IPFS InterPlanetary File System

JSON JavaScript Object Notation

KYC Know Your Customer

MFA Multi-Factor Authentication

NIST National Institute of Standards and Technology

NIZK Schnorr Non-Interactive Zero-Knowledge

NFT Non-Fungible Token

NRO Non-Repudiation of Origin

NRR Non-Repudiation of Reception

NTT Non-Transferable Token

NVS Name-Value Storage

OTP One-Time Password

P2P Peer-to-Peer

PBFT Practical Byzantine Fault Tolerance

PKG Private Key Generator

PKI Public Key Infrastructure

PII Personally Identifiable Information

PoA Proof-of-Authority

PoS Proof-of-Stake

PoT Proof-of-Trust

PoW Proof-of-Work

PPI Privacy-Preserving Identity

RejNFT Rejectable Non-Fungible Token

RejSBT Rejectable Soulbound token

SBT Soulbound token

SSI Self-Sovereign Identity

SSL Secure Sockets Layer

TFS Trust Framework Solutions

TLS Transport Layer Security

TTP Trusted Third Party

UI User Interface

USD US Dollar

USPS United States Postal Service

VC Verifiable Credential

zkSNARK Zero-Knowledge Succinct Non-Interactive Argument of Knowledge

zkSTARK Zero-Knowledge Scalable Transparent Argument of Knowledge

ZKP Zero-Knowledge Proof

zkSBT Zero-Knowledge Soulbound Token

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Blockchain technology

Blockchain has revolutionized the information society since Satoshi Nakamoto published the article *"Bitcoin: A Peer-to-Peer Electronic Cash System"* [1], where a purely Peer-to-Peer (P2P) version of electronic cash is described, using a consensus mechanism to record a public history of transactions that can't be modified. The key innovation of Bitcoin is the use of a distributed computation system, a *Proof-of-Work (PoW)* algorithm, to allow this decentralized network to arrive at a consensus about the state of transactions [2].

Basically, we can consider a blockchain as a decentralized and distributed ledger that records transactions and other data across a network of computers, known as nodes. It enables P2P transactions without the need for intermediaries, such as banks, by allowing participants to directly interact and validate transactions on the network. By leveraging cryptographic techniques and consensus algorithms, blockchain ensures that transactions are secure, transparent, and tamper-proof, promoting trust and eliminating the reliance on centralized authorities.

First blockchains, like Bitcoin or Ripple, were limited to store only value and were improved by a second generation of blockchains like Ethereum and Cardano, that enable decentralized computing thanks to programmable smart contracts. Ethereum, for example, is a deterministic state machine, with a virtual machine that applies changes to that state. It uses blockchain technology to synchronize and store the state changes [3]. In Ethereum, the state represents the current state of the entire network, including all account balances, contract code, and storage. The state is updated through a series of transactions, which are instructions sent by participants on the network. These transactions can involve transferring *Ether (ETH)*, the native cryptocurrency of Ethereum, or executing smart contracts. Every transaction in Ethereum is executed by the nodes on the network in a deterministic manner. This means that given the same initial state and set of transactions, all nodes will produce the same final state. The deterministic nature ensures that the network reaches a consensus on the order and

outcome of transactions.

To achieve determinism, Ethereum uses a virtual machine called the Ethereum Virtual Machine (EVM). The EVM is a runtime environment that executes smart contracts and processes transactions. It provides a sandboxed and isolated environment for the execution of code, ensuring that the execution is predictable and secure.

Blockchain is also bringing us the Internet of value, revolutionizing the way we perceive and engage in financial and business transactions. This groundbreaking technology utilizes cryptography to enable secure and transparent transactions on a global scale. The singularity of blockchain lies in its unique ability to significantly reduce the risk of fraud and hacking throughout the entire history of transactions across various industries. By employing cryptographic techniques, blockchain ensures that each transaction recorded on the decentralized ledger is securely encrypted, making it virtually impossible for unauthorized parties to tamper with or manipulate the data. This inherent immutability and transparency creates a high level of trust and reliability, as every participant in the network can independently verify the integrity of transactions.

Moreover, the decentralized nature of blockchain removes the need for intermediaries, such as banks or clearinghouses, as transactions can be directly executed between parties. This not only streamlines processes but also eliminates the associated costs and delays typically encountered in traditional financial systems.

The impact of blockchain extends beyond financial transactions, encompassing various sectors such as supply chain management, healthcare, real estate, and more. It opens up new possibilities for enhanced efficiency, traceability, and accountability across industries. Through the decentralized consensus mechanisms employed by blockchain, it enables Peer-to-Peer interactions, empowering individuals and organizations to transact directly, bypassing traditional gatekeepers.

With blockchain technology, the possibilities for creating diverse cryptocurrencies are virtually limitless, giving rise to a wide array of tokens that can be utilized in various software solutions. These cryptocurrencies serve as a medium of exchange, enabling secure and efficient transactions within their respective ecosystems, while their ownership and transaction history are securely recorded on the blockchain ledger.

Beyond their function as a means of exchange, cryptocurrencies often possess additional utility within their respective networks. For instance, certain tokens may be utilized as network fees, serving as a form of payment for the computational resources required to process and validate transactions on the blockchain. These fees incentivize network participants, including miners, to contribute their computational power and maintain the decentralized nature of the blockchain network.

Furthermore, cryptocurrencies can act as a reward mechanism, providing an incentive for miners who play a crucial role in securing and validating transactions on the blockchain. These miners dedicate their computational resources to solving complex mathematical puzzles, a process known as mining. In return for their efforts, they are rewarded with newly minted tokens or transaction fees, thus encouraging their ongoing participation in maintaining the integrity and security of the blockchain network.

The ability to create and utilize different types of tokens within blockchain-based software solutions opens up routes for innovative applications and business models [4]. Tokens can represent various assets, such as digital collectibles, real estate holdings, or even voting rights within decentralized governance systems. This tokenization of assets enables fractional ownership, increased liquidity, and opens up new opportunities for

Decentralized Finance (DeFi) applications, where individuals can engage in lending, borrowing, and other financial activities without relying on traditional intermediaries. Instead of the need for intermediaries, these financial transactions are directly made between users, mediated by smart contract programs.

With these characteristics, blockchain technology has been compared to the Internet in the 90s. As Internet 1.0 revolutionized access to information, Blockchain is now revolutionizing access to finances and commerce. This is due to its decentralization and business cost reduction. But blockchain has also other non-finance uses, that will be reviewed in this thesis.

One concept derived from Blockchain technology is **Web3**, which is an idea for a new iteration of the World Wide Web that incorporates concepts such as decentralization, blockchain technologies, and token-based economics, that encapsulates a vision of a more open, trustless, and user-centric internet. The Ethereum co-founder Gavin Wood coined this term, *"Web3"* (originally *"Web 3.0"*), back in 2014.

In the Web3 paradigm, decentralized applications, commonly known as **DApps**, play a central role. These applications are built on blockchain platforms like Ethereum, allowing for the development and deployment of smart contracts that govern their functionalities. *DApps* are designed to operate in a Peer-to-Peer manner, with data and transactions executed directly between users without relying on intermediaries.

To interact with these *DApps*, users employ browser extensions like Metamask. Metamask serves as a digital wallet that not only stores and manages users' cryptocurrency accounts but also acts as a bridge between traditional web browsers and the decentralized world of Web3. It seamlessly integrates with popular web browsers, providing a user-friendly interface and allowing users to securely interact with *DApps*, sign transactions, and manage their digital assets.

The introduction of Web3 and *DApps* brings about numerous possibilities for innovation and disruption across various industries. By leveraging the decentralized nature of blockchain technology, Web3 aims to tackle the challenges of data privacy, security, and control that are prevalent in the traditional Web 2.0 model. It empowers users with ownership and control over their digital identities and data, enabling Peer-to-Peer transactions, and promoting a more inclusive and equitable digital ecosystem.

Moreover, Web3 promotes the growth of Decentralized Finance (DeFi) applications, enabling individuals to participate in various financial activities without relying on traditional financial institutions. It promotes the development of decentralized exchanges, lending platforms, and other financial tools that operate autonomously, transparently, and without the need for intermediaries.

To summarize, blockchain technology possesses several key properties that distinguish it from traditional centralized systems. Here are some prominent properties of blockchain technology:

1. **Decentralization** Blockchain operates as a decentralized network, eliminating the need for a central authority or intermediary to control transactions and data. Instead, multiple participants (nodes) in the network validate and maintain the blockchain's integrity.

2. **Transparency** Blockchain offers transparent access to its data and transactions. Once recorded, information on the blockchain becomes immutable, signifying

that it cannot be altered without consensus from the network participants. This inherent transparency not only promotes trust but also strengthens accountability within the system.

3. **Security** Blockchain employs cryptographic techniques to ensure the security of transactions and data. Each block in the chain contains a cryptographic hash, linking it to the previous block, creating a tamper-proof record. Additionally, consensus mechanisms prevent unauthorized modifications to the blockchain.

4. **Immutability** As mentioned earlier, once information is recorded on the blockchain, it becomes virtually impossible to alter retroactively. This immutability helps maintain data integrity and prevents fraud or manipulation of records.

5. **Distributed Ledger** The blockchain maintains a distributed ledger, which means that copies of the entire blockchain are replicated across multiple nodes in the network. This redundancy enhances fault tolerance, as there is no single point of failure. It also enables all participants to have a synchronized view of the blockchain.

6. **Trust and Consensus** Blockchain employs consensus mechanisms to validate and agree on the state of the blockchain. These mechanisms ensure that all participants reach a consensus on the validity of transactions and maintain the integrity of the blockchain.

7. **Anonymity and Pseudonymity** Blockchain allows users to maintain different levels of privacy. While transactions are visible on the blockchain, the real-world identities behind the transactions are often encrypted or represented by pseudonyms. This feature enhances privacy while ensuring transaction traceability.

8. **Programmable** Blockchain platforms often support **smart contracts**, which are self-executing contracts with predefined rules and conditions. These contracts are automatically enforced and executed when the predetermined criteria are met, eliminating the need for intermediaries and reducing transaction costs.

9. **Interoperability** Blockchain technology can be designed to allow interoperability between different blockchain networks. This feature enables seamless communication and data exchange between disparate blockchain systems, expanding the potential applications and usefulness of the technology.

It's important to note that while these properties generally apply to blockchain technology, specific implementations and variations may differ in certain aspects.

1.2 E-commerce protocols

E-commerce, short for electronic commerce, refers to the buying and selling of goods, services, or information over the Internet. It involves conducting commercial transactions online, typically through websites or online platforms. E-commerce encompasses

a wide range of activities, including online retail, online auctions, online ticketing, digital downloads, online banking, certified notifications, contract signing, and more.

In e-commerce, businesses or individuals use digital platforms to showcase their products or services, facilitate transactions, and interact with customers. Customers can browse through product catalogs, compare prices, make purchases, and make payments electronically. E-commerce has transformed the way businesses and consumers engage in commerce, offering convenience, accessibility, and a global reach.

There are different types of e-commerce models, including:

1. **Business-to-Consumer (B2C)**: This model involves businesses selling products or services directly to individual consumers. Examples include online retail stores, where customers can purchase items from various sellers through a website.

2. **Business-to-Business (B2B)**: In this model, businesses engage in e-commerce transactions with other businesses. It typically involves the exchange of goods, services, or information between companies. B2B e-commerce often involves large-scale transactions, supply chain management, and Electronic Data Interchange (EDI).

3. **Consumer-to-Consumer (C2C)**: C2C e-commerce enables individuals to engage in commerce with each other. Online marketplaces or platforms facilitate these transactions, allowing individuals to buy and sell products or services directly to other individuals. Popular examples include online classifieds or auction websites.

4. **Consumer-to-Business (C2B)**: C2B e-commerce occurs when individuals offer products, services, or expertise to businesses. This model is commonly seen in freelance platforms or websites where individuals can provide services such as graphic design, writing, or consulting to businesses.

5. **Business-to-Government (B2G)**: B2G e-commerce involves businesses engaging in online transactions with government entities. This can include businesses bidding on government contracts or providing goods and services to government agencies through digital platforms.

E-commerce has experienced significant growth due to advancements in technology, widespread internet access, and evolving consumer preferences. It offers benefits such as convenience, 24/7 availability, access to a global customer base, reduced costs, and personalized shopping experiences.

There are various operations associated with e-commerce protocols that extend beyond the typical ones like buying/selling or payments. These include confidential and non-confidential notifications, contract signing, micropurchases, and digital identity management. Let's look at each of them in more detail:

1. **Confidential and Non-Confidential Notifications** (or certified eDeliveries): involve sending information securely between parties involved in an e-commerce transaction. This can include notifications of order status, shipment updates, payment confirmations, or any other communication related to the transaction. These notifications can be public (non-confidential) or private (confidential).

In the case of confidential notifications, ensuring the confidentiality of these notifications is crucial to maintaining privacy and security in e-commerce.

2. **Contract Signing**: Contract signing involves the application of cryptographic methods to validate the authenticity and integrity of digital contracts or agreements. In e-commerce and business transactions, contract signing serves to authenticate the identity of the signatories and guarantee the preservation of data integrity throughout transmission. This process offers confidence that the signing parties are genuine and that the contract content has remained unaltered.

3. **Micropurchases**: Micropurchases refer to small-value transactions conducted in e-commerce, typically involving low-cost products or services. Micropurchases are often characterized by their simplicity, speed, and convenience. They can include quick online purchases of digital goods, small physical items, or services with minimal financial commitment.

4. **Digital Identity Management**: Digital identity management refers to the process of managing and controlling the online representation of an individual or entity. In e-commerce, digital identity management plays a crucial role in establishing trust and verifying the identities of users involved in transactions. Digital identity management systems focus on validating the attributes of user identities, ensuring their authenticity and security in e-commerce transactions.

While these operations are not exclusive to e-commerce protocols, they are often utilized within e-commerce platforms and systems to enhance security, privacy, convenience, and trust in online transactions.

1.3 Blockchain technology applied to e-commerce protocols

Blockchain technology can be applied to e-commerce protocols and has the potential to bring several advantages to the e-commerce industry. Here are some ways in which blockchain can be used in e-commerce:

1. **Enhanced Security**: Blockchain's decentralized and cryptographic nature provides a higher level of security for e-commerce transactions. It can help protect sensitive customer information, prevent fraud, and ensure the integrity of data.

2. **Transparent and Trustworthy Transactions**: Blockchain's transparency enables customers and businesses to have greater trust in the e-commerce process. By recording transaction details on the blockchain, customers can verify the authenticity of products, track the supply chain, and ensure fair pricing.

3. **Streamlined Payments**: Blockchain-based cryptocurrencies, such as Bitcoin (BTC) or Ether (ETH), can simplify and speed up payment processes in e-commerce. By eliminating the need for intermediaries like banks or payment processors, blockchain-based payments can reduce transaction fees and enable faster cross-border transactions.

4. **Smart Contracts for Automation**: Smart contracts on the blockchain can automate various aspects of e-commerce, such as order fulfillment, payment releases, and dispute resolution. These self-executing contracts can streamline and expedite transactions while reducing reliance on intermediaries.

5. **Decentralized Marketplaces**: Blockchain technology can enable the development of decentralized e-commerce platforms. These platforms allow buyers and sellers to interact directly, eliminating the need for a central authority. They can also provide features like reputation systems, dispute resolution, and transparent feedback mechanisms.

6. **Supply Chain Transparency**: Blockchain can enhance supply chain management in e-commerce by providing a transparent and immutable record of the entire supply chain process. This enables customers to trace the origin and authenticity of products, ensuring ethical sourcing and reducing counterfeiting.

7. **Loyalty Programs and Customer Rewards**: Blockchain-based loyalty programs can provide customers with more transparent and flexible rewards systems. By tokenizing loyalty points on the blockchain, customers can easily track, transfer, and redeem their rewards across different merchants.

8. **Data Privacy and Ownership**: Blockchain technology can give customers more control over their personal data by enabling secure and permissioned access. Customers can choose which information to share and with whom, reducing the risks of data breaches and unauthorized use of personal data.

While blockchain technology holds promise for e-commerce, it's important to consider factors like scalability, user experience, and regulatory compliance when implementing blockchain solutions in this domain.

In the current landscape of e-commerce, trust and security are paramount. Blockchain's decentralized architecture eliminates the need for a central authority, reducing the risk of fraud and ensuring transparency in transactions. The use of smart contracts, self-executing contracts with coded terms, further enhances the efficiency of e-commerce processes by automating and enforcing contractual agreements.

Several existing blockchain-based e-commerce applications and protocols showcase the technology's potential. From supply chain management to payment processing, blockchain introduces innovative solutions that streamline operations and enhance the overall integrity of transactions. Notable examples include traceability of goods, enabling consumers to verify the authenticity of products, and the facilitation of cross-border transactions with reduced intermediary involvement.

Despite these advancements, challenges persist, and the adoption of blockchain in e-commerce is not without hurdles. Issues such as scalability, interoperability, as well as security properties like privacy, efficiency, and integrity, need careful examination. The thesis aims to delve into these challenges, proposing solutions and contributing to the evolving body of knowledge surrounding the effective integration of blockchain in e-commerce protocols.

By examining the current state of blockchain in e-commerce, this research seeks to identify gaps and opportunities for improvement. The analysis will not only focus

on protocol definitions but also provide a balanced perspective on the technology's viability. As blockchain continues to evolve, its integration into e-commerce protocols has the potential to redefine the way online transactions are conducted, promoting a more secure, transparent, and efficient digital marketplace.

1.4 Structure of this thesis

The thesis is organized into a logical progression of chapters, beginning with the introductory and foundational aspects and culminating in a comprehensive exploration of blockchain technology's applications in e-commerce. Chapter 1, Chapter 2 and Chapter 3 lay the groundwork for the thesis, encompassing the introduction, which outlines the research problem, objectives, and significance, followed by a detailed exposition of the research methodology. These initial chapters are crucial for setting the context and providing the reader with a clear understanding of the study's aims and approach.

Chapter 4 stands alone as an in-depth examination of blockchain technology, providing the necessary background information and technical foundation. This chapter also addresses the evaluation of blockchain-related technologies in practical e-commerce scenarios. Chapter 5 transitions into specific use-cases of blockchain in e-commerce, demonstrating the technology's potential and challenges, while Chapter 6 further delves into the development and implementation aspects.

The main body of the thesis, from Chapter 8 to Chapter 16, is dedicated to blockchain protocols and their applications within e-commerce. This section forms the core of the research, presenting detailed investigations, applications, and evaluations of various blockchain protocols and their integration into e-commerce systems. It is here that the thesis makes its most significant contributions, detailing the design, implementation, and impacts of blockchain-based solutions. It includes the detailed description of the following protocols: rejectable NFTs (Chapter 7), two-party certified notifications (Chapter 8), multiparty non-confidential and confidential (using a TTP) certified notifications (Chapter 9), confidential certified notifications without TTP (Chapter 10), improved certified notifications protocol (Chapter 11), two-steps certified notifications (Chapter 12), two-party contract signing (Chapter 13), confidential multiparty contract signing (Chapter 14), micropurchases using payment channels (Chapter 15), and decentralized and confidential digital identity management system using Zero-Knowledge Proof (ZKP) (Chapter 16).

The concluding section, Chapter 17, wrap up the thesis with a presentation of the research results, a comprehensive discussion linking the findings back to the research objectives, and a conclusion that summarizes the study's main contributions and implications. This final chapter also outlines the limitations of the current research and suggests routes for future work, providing a clear path forward for subsequent investigations in the field.

Through this structured approach, the thesis offers a cohesive and thorough examination of blockchain technology in the e-commerce domain, from foundational concepts to advanced applications and critical evaluations, ensuring a comprehensive understanding of the subject matter for the reader.

CHAPTER 2

OBJECTIVES

The main objective of this research project is to study and create different blockchain-based protocols applied to e-commerce, emphasizing the need for robust security mechanisms. The research aims to achieve enhanced characteristics of security, privacy, and trust through a distributed structure enabled by blockchain technology. Unlike traditional structures that rely on the intervention of a centralized TTP, the proposed approach seeks to minimize the need for trust in third parties and instead leverage the decentralized nature of blockchain.

By adopting a distributed structure, the research project intends to increase effectiveness and efficiency while reducing reliance on centralized authorities. Technologies based on blockchain will be evaluated to check their suitability in achieving the desired outcomes for applications that require secure and private interchange operations. These applications encompass various areas such as fair exchanges, notifications, contract signatures, payment systems, and digital identity management.

The underlying principle of fair interchange in these applications is to ensure that all users receive fair and egalitarian treatment. At the conclusion of each execution, the aim is for each party involved to have obtained the desired element or to determine that the exchange has not been completed correctly. This equitable approach aims to create a balanced and just system that eliminates bias and promotes equal participation among users.

Taking all of this into consideration, the general objectives of this thesis are:

1. To analyze the limitations and challenges of traditional e-commerce protocols: This objective involves identifying the shortcomings of existing e-commerce protocols, such as security vulnerabilities, lack of transparency, scalability issues, or dependence on TTPs.

2. To explore the potential of blockchain technology in enhancing e-commerce protocols: This objective aims to investigate how blockchain can address the identified limitations of traditional e-commerce protocols. It involves under-

standing the unique properties of blockchain, such as decentralization, transparency, immutability, and security, and assessing how they can be leveraged to improve e-commerce transactions.

3. To design and create a set of blockchain-based e-commerce protocols: This objective involves designing new e-commerce protocols that incorporate blockchain technology. The focus is on creating protocols that take advantage of blockchain's features to enhance security, transparency, efficiency, and trust in e-commerce transactions.

4. To evaluate the performance and effectiveness of these blockchain-based e-commerce protocols: This objective entails assessing the performance of the developed protocols by evaluating factors like scalability, transaction speed, transaction costs, and security. It may involve conducting experiments, simulations, or user studies to gather data and analyze the protocol's effectiveness.

By leveraging the security, transparency, and decentralized nature of blockchain technology, the research project seeks to enable secure and private interactions in e-commerce. The goal is to develop protocols that can provide robust security mechanisms, protect privacy, and establish trust without relying on centralized authorities. Ultimately, the successful implementation of these protocols will contribute to the advancement of fair and efficient interchange operations, paving the way for enhanced e-commerce experiences and facilitating fair and secure transactions for all participants involved.

CHAPTER 3

METHODOLOGY

The methodology employed in this thesis on blockchain-based e-commerce protocols will involve several key steps to ensure a comprehensive and effective approach. It will start with an extensive review of existing literature and prior research in the field. This literature review will provide a solid foundation and understanding of the current state-of-the-art, technologies, and challenges associated with blockchain-based e-commerce protocols.

Based on the insights gained from the literature review, the next step will be to generate a list of requirements that the proposed protocols should meet. These requirements will encompass various aspects such as security, privacy, scalability, interoperability, and usability, among others. This step is crucial in defining the scope and objectives of the research project and setting the criteria against which the proposed protocols will be evaluated.

Following the establishment of requirements, the research will progress to the design and development of new protocol proposals. These proposals will aim to address the identified requirements and overcome the limitations of existing protocols. The design phase will involve the formulation of new approaches, algorithms, and mechanisms that can enhance security, privacy, and overall efficiency in e-commerce transactions conducted on the blockchain.

Subsequently, the proposed protocols will undergo a thorough analysis of their security aspects. This analysis will involve evaluating the protocols against potential vulnerabilities, threats, and attack vectors. Any identified weaknesses or areas for improvement will be addressed, and appropriate countermeasures will be developed to enhance the security of the protocols.

Once the protocols have been refined and improved, the next step will involve their implementation and performance testing. The implementation phase will bring the proposed protocols to life, transforming the designs into functional code that can be tested and evaluated. The performance tests will measure the efficiency, scalability, and robustness of the protocols under various scenarios and workloads.

In the final stage, efforts will be made to disseminate and transfer the knowledge

gained from the research to interested companies or entities. This may involve collaborating with industry partners, presenting research findings at conferences or workshops, and publishing academic papers. The aim is to bridge the gap between academic research and practical applications by sharing valuable insights, techniques, and protocols with relevant stakeholders who can benefit from the materials.

Overall, this methodology ensures a systematic and rigorous approach to the research project, encompassing literature review, requirements generation, protocol design, security analysis, implementation, performance testing, and knowledge transfer. By following this methodology, the research aims to contribute to the advancement of blockchain-based e-commerce protocols and provide valuable insights and solutions to the broader e-commerce community.

CHAPTER 4

BLOCKCHAIN AND RELATED TECHNOLOGIES AND STANDARDS

A blockchain is a distributed database, made up of a chain of blocks that are designed to prevent modification once a piece of data has been published. This is achieved by linking these blocks to their previous block through the cryptographic hash of the previous block. Blocks also contain a timestamp and transaction data. With that, blockchains are specially used for storing data in an orderly manner over time and without the possibility of modification.

Blockchains are also considered as a distributed computing system with high Byzantine fault tolerance, being managed by a Peer-to-Peer (P2P) network, where its nodes collaborate to communicate and validate new blocks.

Blockchain technology is used to solve the problem of trust through cooperation to achieve common goals. It uses advanced mathematics, cryptography, programming languages, and distributed ledger technology.

4.1 Components of a blockchain

The common components of a blockchain are

- **Transactions** that represent the messages that change the state of the data in the blockchain.
- A **state machine** that stores the data and processes transactions. This information is stored in blocks.
- A **hash function** that secures the information stored in the blocks, preventing its modification. Each block includes the hash of the prior block, confirming the integrity of the previous block.

- A **Peer-to-Peer (P2P) network** that propagates transactions and blocks. The nodes of this network broadcast the transactions that pretend to be included in the blocks.

- A **consensus rule** like *Proof-of-Work (PoW)*, *Proof-of-Stake (PoS)*, or *Proof-of-Authority (PoA)* that controls who can add blocks to the chain. In *PoW* consensus, each node of the blockchain competes to solve a guessing game problem. *PoS* consensus attempts to overcome scalability concerns imposed by *PoW* consensus, removing the guessing game from consensus. In *PoS* the validator is selected in proportion to their quantity of holdings in the associated cryptocurrency. In *PoA* consensus there are approved accounts, known as validators, that validate transactions and blocks.

These components are typically combined in client software, like Bitcoin Client, Go Ethereum (*geth*), or the Parity Ethereum Client.

With that, we can see that the term "Blockchain" not only refers to the decentralized ledger that stores information or the Peer-to-Peer network of nodes, but it also refers to the rules and protocols that manage this system.

4.2 Properties of the blockchain

Once we have defined what is a blockchain and what components it has, we will analyze its properties.

A blockchain is **decentralized** because we store data across its Peer-to-Peer network (see Figure 4.1). It doesn't have a single governing authority looking after the framework. One of the key benefits of decentralization is that we eliminate a central point of vulnerability, ensuring its security. But we can have the risk of a "51% attack", where an entity can manipulate a specific blockchain if it gains control of more than half of a network.

In the blockchains, all the nodes have a copy of the information, replicating the entire database, and transactions are broadcasted to the network. The nodes validate the transactions, add them to a new block, and broadcast this new block to the rest

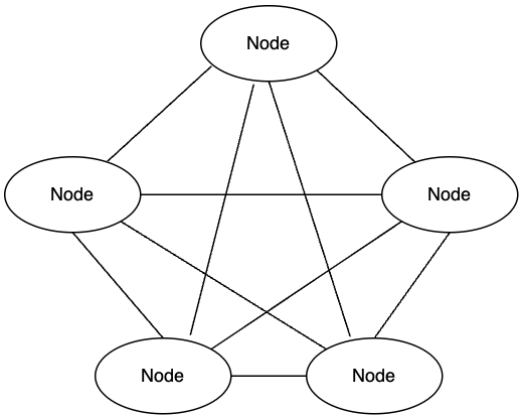


Figure 4.1: Blockchain Peer-to-Peer network

of the nodes. To serialize changes, the nodes use various time-stamping schemes like *Proof-of-Work* or *Proof-of-Stake* to serialize changes [5].

A blockchain is also **immutable** because its information can't be changed or altered. This is accomplished with the use of a hash function that secures the information stored in the blocks. With that, no one can go back to a previous block and change the information, because the hash value won't be valid. This prevents the edition and deletion of information.

Another fact that helps to the inalterability of the information is the need for the validation of most of the nodes of the blockchain. To validate a new transaction every node needs to check its correctness. And if most of the nodes validate the transaction, then the information is added to the ledger. With that, we can't add new information without consent from most of the nodes of the network.

Blockchain technology is more **secure** than other database systems. This is because transactions are added only when the majority of the nodes of the network validate the information, with a consensus system. If the block is validated, it is encrypted and linked with the previous block. A hash mechanism is used to secure and link the information stored in the blocks. With that, every block includes the hash of the prior block (see Figure 4.2), making it very difficult to modify a block as it requires altering the information in other blocks in the chain too.

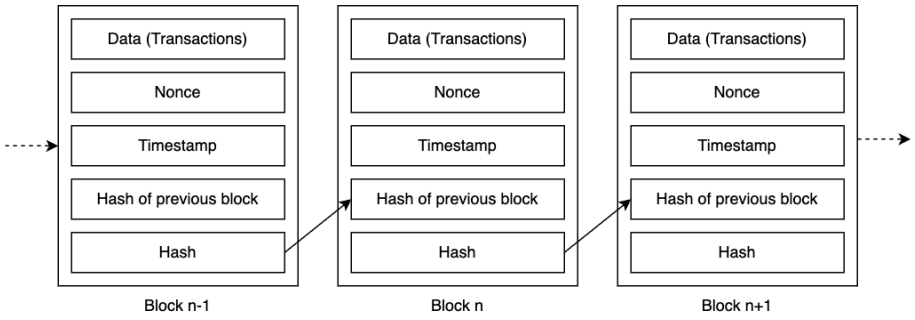


Figure 4.2: Simplified blockchain diagram

Other important properties of this technology are **traceability** and **auditability**. This is due to the ease of tracking the transactions because they are visible to all parties. Blockchain incorporates accounting mechanisms that let it to a traceable source of truth for transaction evidence. We can consult the information of the blockchain transactions and check its veracity because everyone can read this information because we also have the **transparency** property. Taking advantage of these properties, for example, Zou et al. [6] propose a new *Proof-of-Trust (PoT)* consensus protocol, that uses blockchain as the underlying technology to enable tracing transactions for service contracts and dispute arbitration. The traceability property is also used in different articles, to be applied to complex manufacturing systems [7], general supply chain [8], mineral supply chain [9], fruit and vegetable agricultural products [10], etc...

These core properties collectively contribute to the robustness and reliability of blockchain technology, as summarized in Table 4.1.

Table 4.1: Summary of blockchain properties

Property	Description
Decentralization	Distribution of control among network participants.
Immutability	Inability to alter data once recorded on the blockchain.
Security	High level of protection against tampering and unauthorized access.
Traceability and Auditability	Ability to trace and audit transactions for accountability.
Transparency	Openness and visibility of transactions to all network participants.

4.3 Types of blockchain networks

Depending on the access type of the users of the blockchain, we can have four different types of blockchains.

First of all, a blockchain can be **public** or **permissionless** [11], if it has no access restrictions. In these types of blockchains, anyone can send transactions to it or become a miner who can validate the transactions. Normally, these networks use consensus mechanisms that offer economic incentives to their miners, like *Proof-of-Work* and *Proof-of-Stake*. The largest blockchains, like Ethereum or Bitcoin, are of this type.

Blockchains can also be **private** or **permissioned** [12], where only certain participants have write and read permissions. In these kinds of networks, the nodes cannot join unless invited by the network administrators. Hyperledger, for example, is a permissioned and private blockchain infrastructure, which can give different roles to the participants of this network. Hyperledger can use different consensus algorithms, like *Practical Byzantine Fault Tolerance (PBFT)* [13]. These kinds of networks can be applied to different use cases, like securing medical forensic systems [14], supply chain finance [15] and multistage quality control and enhancing security in smart manufacturing [16]. Sometimes, to refer to private blockchain we use the terminology Distributed Ledger Technology (DLT).

We can also use **hybrid** blockchains that have a combination of centralized and decentralized features [17]. Depending on the network, we can find different portions of centralization and decentralization properties.

There are also **sidechains**, which are secondary blockchain ledgers that run in parallel to the main blockchain [18]. They can have their own consensus protocol and can have different properties to add new functionalities, to improve the scalability, privacy, and security of the main blockchain. Sidechains can also operate independently of the primary blockchain.

4.4 Ethereum blockchain

Ethereum is considered a Blockchain 2.0 technology. It's an open-source Blockchain network that can act as a decentralized *Turing-complete* Virtual Machine [19]. Its native token is *Ether (ETH)*, which is also used to perform payments, in the form of **gas**, for executing computer programs [20].

One of the innovations of Ethereum is the use of **smart contracts** [21], that store computer code written in multiple languages like Solidity or Vyper. This code is executed on the **Ethereum Virtual Machine (EVM)**, which can run **Ethereum Bytecode**. EVM follows the EVM specifications, written in the Ethereum protocol, and runs as a process on a computer. The EVM is implemented in different programming languages.

Ethereum is not only a state machine that tracks the transitions of the state of currency ownership, like Bitcoin. Ethereum also tracks the state of a general-purpose data store. In this data store, it can be stored both code and data, and the data can be any data expressible as a key-value tuple. This blockchain can act as a general-purpose computer, running the smart contract's code, and storing the results. But this distributed computer uses a consensus system to govern the state changes and the data is stored globally.

In Ethereum, we have two types of accounts: *Externally-Owned Accounts (EOAs)*, which represent external users and are controlled by a private key, and *Smart Contract Accounts*. Both types of accounts can store *Ether (ETH)* balance and both types of accounts can create new smart contracts or call any public function of a smart contract. Additionally, *Smart Contract Accounts* can store code, which defines the logic and functionality of the smart contract.

All the accounts are identified by their address. In the case of the EOA accounts, they are composed of the prefix "0x" and the rightmost 20 bytes of the *Keccak-256* hash of the *ECDSA* public key of that account. The public key is calculated from the private key using elliptic curve multiplication, as defined in a standard *secp256k1*, and the private keys are a 64-character hexadecimal string. EOA accounts are the only type of accounts that can create transactions that need to be signed by the account's private key. It's important to notice that anyone can derive the signer's address from the signature without knowing the private key.

On the other hand, smart contract addresses have the same format as the EOA or users' accounts, but they are determined by the creator's address and the transaction nonce that creates it. As mentioned earlier, smart contracts are the only type of accounts with associated code.

There are mechanisms like **Ethereum Name Service (ENS)** to translate account addresses to human-readable names with a dot-separated right-to-left hierarchical naming structure [3]. It works similarly to the *Domain Name System (DNS)* in the traditional Internet but in a decentralized manner.

4.5 Scalability problems in Ethereum

The Ethereum network has some scalability issues, affecting the efficient use of the network in a massive way, being waiting time and transaction fees as the most problematic ones. These problems have become more evident with the emergence of DeFi protocols and applications.

Ethereum underwent a significant transformation with the launch of Ethereum 2.0, transitioning from the *Proof-of-Work (PoW)* to the *Proof-of-Stake (PoS)* consensus mechanism. This upgrade has vastly improved the platform's energy efficiency, reduced hardware requirements, and facilitated the proliferation of network nodes. Additionally, Ethereum 2.0 introduced robust support for *shard chains*, which distribute the network's

workload across 64 new chains, greatly enhancing scalability and capacity. *Shard chains* are individual chains within the Ethereum network that process and store specific subsets of transactions, allowing for parallel processing and significantly increasing the overall throughput of the network. While this marks a substantial step forward in addressing Ethereum's scalability issues, it's important to note that the merge to Ethereum 2.0 was successfully completed in September 2022.

Many articles, like [22, 23, 24], try to suggest different approaches to the scalability problem. These articles describe the scalability problem, caused by the typical blockchain design, that requires every node in the network to process every transaction. With that, the capacity of a single node limits the transaction processing capacity of the entire system.

There are other solutions, taking advantage of a *second layer*, that operate on the native layer to improve its performance. The *second layer* takes a portion of the native or first layer's transactions, offloading them, and giving it to another system architecture. With that, the *second layer* handles the processing load and informs the first layer for result finalization, reducing network congestion.

In [25] it's proposed a scalability solution with *rollups*, a *second layer* scaling paradigm. This solution performs transaction execution outside the main layer but posts transaction data on it. As transaction data is on the first layer, *rollups* are secured by this layer. There are many kinds of *rollups*, for example, ones using fraud proofs, or *ZK rollups* using validity proofs.

4.6 Token standards

Token standards are fundamental to the functionality and interoperability of digital assets within the blockchain technology ecosystem. These standards, which are predominantly established within the Ethereum network, provide a uniform protocol for the creation, transaction, and management of tokens. In this context, ERC stands for Ethereum Request for Comment, denoting a standardized protocol within the Ethereum ecosystem. This section introduces the token standard such as **ERC-721** or Non-Fungible Token (NFT), and further explores innovative concepts such as Soulbound tokens (SBTs).

Tokens represent digital assets exchanged and operated within the blockchain ecosystem, significantly enhancing the security, efficiency, and scalability of e-commerce transactions. Standards such as **ERC-20** and **ERC-721** facilitate the creation and management of fungible and non-fungible tokens, respectively. The **ERC-20** standard [26] enables the creation of interchangeable tokens where each token holds the same value, thus ideal for creating cryptocurrencies. On the other hand, the **ERC-721** standard [27] defines the necessary specifications for Non-Fungible Tokens (NFTs), which represent unique digital properties and cannot be interchanged directly with other tokens due to their distinct characteristics. Additionally, Soulbound tokens, which are tokens permanently or semi-permanently tied to a wallet, pave the way for innovative applications in identity verification and beyond, and will be used in the protocols described in Chapter 12 and Chapter 16.

By leveraging these token standards, the thesis aims to develop blockchain-based e-commerce protocols that address the challenges faced by traditional systems while

promoting an interconnected digital economy. The standardized approach ensures consistency and compatibility across multiple platforms and applications, laying the foundation for a robust and versatile e-commerce ecosystem.

4.6.1 EIP-721: Non-Fungible Token Standard

EIP-721 represents a standard interface for Non-Fungible Tokens [28], providing basic functionality to track and transfer them, letting us track the ownership of each one separately. Non-Fungible Tokens (NFTs) can represent ownership of distinguishable digital or physical assets and the right of possession is recorded in the blockchain. The owner of the NFT is represented by its wallet address. However, the NFT transfer cannot only be initiated by its owner but also by its approved address or even by an authorized operator of the current owner of the NFT.

This standard has the following events (see Listing 1):

- `Transfer`: emitted when any NFT changes its ownership. This is also emitted when the NFT is created and destroyed.
- `Approval`: emitted when the approved address for an NFT is changed or reaffirmed. It's also emitted when the **Transfer** event is emitted and the approved address for that NFT is reset to none, with the approved address set to the zero address.
- `ApprovalForAll`: emitted when an operator is enabled or disabled for an owner. Operators can manage all NFTs of the owner.

```
event Transfer(address indexed _from, address indexed _to,
  uint256 indexed _tokenId);

event Approval(address indexed _owner,
  address indexed _approved, uint256 indexed _tokenId);

event ApprovalForAll(address indexed _owner,
  address indexed _operator, bool _approved);
```

Listing 1: ERC-721 Events.

In addition, the EIP-721 standard has the following functions (see Listing 2):

- `balanceOf(address _owner) returns (uint256)`: Returns the number of non-fungible tokens owned by the specified owner.
- `ownerOf(uint256 _tokenId) returns(address)`: Returns the address of the owner of the NFT with the specified Id.
- `safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data)`: Transfers the ownership of an NFT from one address to another address. This function also checks if *_to* parameter is a smart contract ($codesize > 0$).
- `safeTransferFrom (address _from, address _to, uint256 _tokenId)`: Same as previous function, with *data* parameter set to blank string ("").

- `transferFrom(address _from, address _to, uint256 _tokenId)`: Transfers the ownership of an NFT from one address to another address, without checking if *_to* parameter is a smart contract.
- `approve(address _approved, uint256 _tokenId)`: Changes or reaffirms the approved address for an NFT. If *_approved* is set to the zero address, that indicates that there is no approved address.
- `setApprovalForAll(address _operator, bool _approved)`: Enables or disables approval for a third party "operator" to manage all of the caller of the function assets.
- `getApproved(uint256 _tokenId) returns (address)`: Gets the approved address for a single NFT.
- `isApprovedForAll(address _owner, _operator) returns (bool)`: Queries if an address is an authorized operator for another address.

```
function balanceOf(address _owner) external view
  returns (uint256);

function ownerOf(uint256 _tokenId) external view
  returns (address);

function safeTransferFrom(address _from, address _to,
  uint256 _tokenId, bytes data) external payable;

function safeTransferFrom(address _from, address _to,
  uint256 _tokenId) external payable;

function transferFrom(address _from, address _to,
  uint256 _tokenId) external payable;

function approve(address _approved, uint256 _tokenId)
  external payable;

function setApprovalForAll(address _operator,
  bool _approved) external;

function getApproved(uint256 _tokenId) external view
  returns (address);

function isApprovedForAll(address _owner,
  address _operator) external view returns (bool);
```

Listing 2: ERC-721 Functions

4.6.2 Soulbound tokens

Soulbound tokens (SBTs) are a type of digital asset that is designed to be unique, non-fungible, and tied to a specific individual. They are also called Non-Transferable Non-Fungible Tokens (NFTs). This special type of token was introduced by Weyl et al. in

[29]. They are created using blockchain technology and are often used as a means of verifying and managing digital identities in a self-sovereign manner.

Soulbound tokens can be used in Self-Sovereign Identity (SSI) systems (see section 5.4) as a way of establishing and verifying an individual's identity. When authorities create a Soulbound token, they are essentially creating a digital representation of something that is unique and tied to a specific identity. This token can be used to authenticate the individual's identity when interacting with other parties, such as financial institutions, government agencies, or online service providers.

Because Soulbound tokens are based on blockchain technology, they are highly secure and difficult to tamper with. This makes them an ideal tool for managing digital identities in a self-sovereign manner, as they provide a high degree of security and privacy while still allowing individuals to maintain control over their personal data.

There exists EIPs like EIP-4671 [30], EIP-5114 [31] or EIP-5484 [32] that describe a Soulbound token. In any case, we must bear in mind that these EIPs are under constant review. This kind of token is attached to a "soul" at mint time and cannot be transferred after that. Typically this "soul" is another NFT, thus a collection of NFTs can be linked to a single NFT to guarantee non-separability and non-mergeability of the token collection. With this standard, we have tokens that cannot be sent to another account because they are attached to a token but, in this thesis, we will propose tokens that could only be sent with the approval of the receiver.

To develop the SBTs of the protocol described in Chapter 16 we will use the Non-Transferable Tokens (NTTs), standardized by EIP-4671[30], which facilitates interoperability and seamless integration of SBTs across diverse blockchain ecosystems. Notably, EIP-4671 creates a new interface, and it doesn't extend the ERC-721 standard. Another interesting capability of NTTs is the faculty of the deliverer of these tokens to revoke it, but not to delete it [33]. This fosters a more coherent and cohesive identity infrastructure, empowering users with enhanced control and security over their digital identities.

4.7 Cryptographic technologies used in blockchain

Cryptographic technologies serve as the backbone of security and privacy in blockchain-based e-commerce protocols. Public-key cryptography, symmetric-key cryptography, and cryptographic hashing algorithms are utilized to safeguard sensitive information, secure transactions, and ensure data integrity. Through the use of cryptographic techniques, such as digital signatures and encryption, we can authenticate users, protect their identities, and enable secure communication within the e-commerce protocol. Additionally, cryptographic hash functions play a critical role in verifying the integrity of data and ensuring its immutability on the blockchain. In this section, we will describe the cryptographic technologies that are fundamental to our discussion and will later be applied to elaborate specific protocols within this thesis on blockchain-based e-commerce protocols.

4.7.1 Hash functions

Hash functions play a crucial role in blockchain technology, serving as the foundation for ensuring security, integrity, and efficiency within the blockchain ecosystem. A hash

function is a mathematical algorithm that takes input data of any size and produces a fixed-size output, known as a hash value or hash code. This output is unique to the specific input data, meaning even a minor change in the input will result in a completely different hash value.

In blockchain technology, hash functions serve several essential purposes. One primary use is in the creation of digital signatures. Hash functions allow users to create a unique fingerprint of a digital document or transaction, which can be securely and efficiently verified without revealing the original content. This property makes hash functions invaluable for ensuring the authenticity and integrity of data within the blockchain.

Another critical use of hash functions is in the creation of blocks in the blockchain. Each block contains a unique hash value that represents the entire content of that block, including the transactions it contains and the hash value of the previous block. By linking blocks together using hash values, the blockchain creates an immutable and tamper-resistant ledger. Any alteration of the data within a block would result in a different hash value, thereby breaking the chain and immediately alerting participants of the tampering attempt.

Moreover, hash functions contribute to the efficiency and security of blockchain networks. They allow for quick verification of data integrity, as comparing hash values is computationally efficient. Furthermore, hash functions are designed to be one-way, meaning it is practically impossible to reverse-engineer the original input data from its hash value. This property protects sensitive information and ensures the privacy and security of participants within the blockchain.

Overall, hash functions are a fundamental building block of blockchain technology. They provide the cryptographic foundations necessary for secure and reliable data storage, transaction validation, and consensus mechanisms within a decentralized network. The use of hash functions in blockchain technology ensures trust, transparency, and immutability, enabling a wide range of applications such as cryptocurrency, smart contracts, supply chain management, and more.

4.7.2 Merkle trees

The Merkle tree is a fundamental data structure used in blockchain technology that provides efficient and secure verification of data integrity. It was named after Ralph Merkle, who invented the concept in the late 1970s. The Merkle tree's unique properties make it an essential component of many blockchain systems, including Bitcoin and Ethereum.

The Merkle tree, also known as a hash tree, is a binary tree structure where each leaf node represents a data block or transaction (see Figure 4.3). The parent nodes in the tree are created by hashing the concatenation of their child nodes' hashes. This process continues until a single root hash, known as the Merkle root, is obtained at the top of the tree.

One of the key benefits of a Merkle tree is its ability to efficiently verify the integrity of a large dataset without requiring access to all the individual data elements. By comparing a small subset of hash values—typically logarithmic in the total number of data elements—a participant can ensure that the data has not been tampered with.

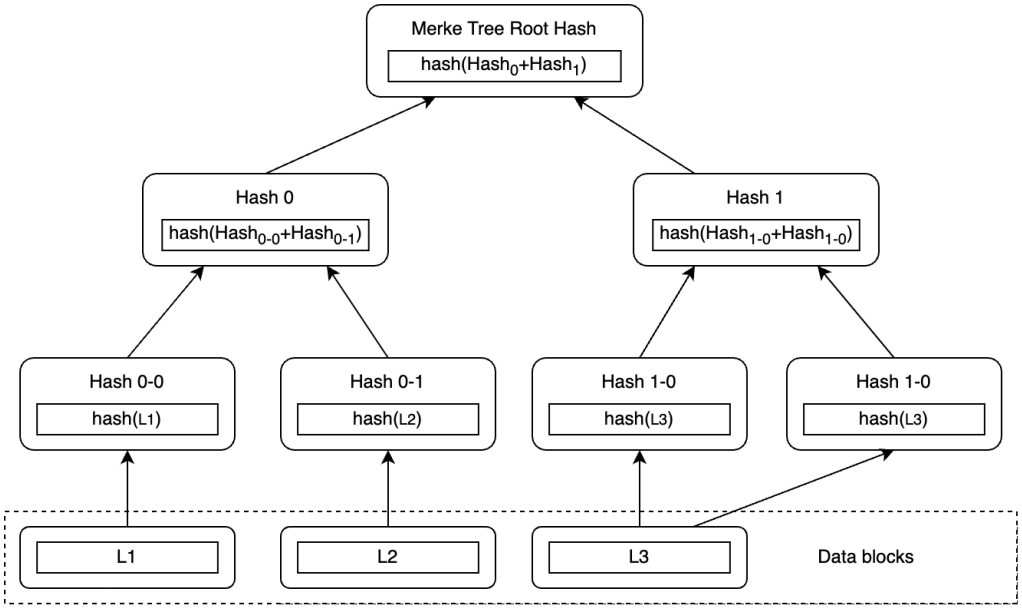


Figure 4.3: Merkle tree structure

This property makes the Merkle tree ideal for use in blockchain systems, where data integrity and security are paramount.

In the context of blockchain technology, Merkle trees are extensively used in various ways. One of the prominent applications is in the efficient verification of transactions within a block. Instead of validating each transaction individually, which can be computationally expensive, a participant can simply validate the Merkle root against a known value. If the Merkle root matches the expected value, it guarantees the integrity and inclusion of all the transactions in that block.

Furthermore, Merkle trees play a crucial role in ensuring the security and immutability of the blockchain. Since the Merkle root is included in the block header, any change in the underlying data would result in a different Merkle root. This property enables easy detection of any tampering or unauthorized modifications to the blockchain data.

Moreover, Merkle proofs, which are generated using the Merkle tree structure, enable efficient and compact verification of individual transactions. By providing a series of hash values and their corresponding sibling nodes, a participant can prove the inclusion or absence of a specific transaction without needing the entire block data.

In addition to Merkle trees, Ethereum also utilizes a more complex structure known as a *Merkle Patricia Trie*. This data structure combines the advantages of a Merkle tree with those of a Patricia trie, enabling more efficient searches and updates. A *Merkle Patricia Trie* differs from a traditional Merkle tree in that it provides a path to every inserted value based on its key, significantly optimizing the process of looking up and updating values. This is especially beneficial in Ethereum, where it is used to store all the state information, including accounts, balances, and storage. The *trie* structure ensures that every state transition results in a new, unique root hash, thus maintaining a historical record of all changes in a compact form.

In summary, the Merkle tree and the *Merkle Patricia Trie* are foundational compo-

nents in blockchain technology, providing efficient and secure data integrity verification. Their ability to reduce computational overhead, detect tampering, and enable compact proofs makes them integral parts of blockchain systems, contributing to the trust and reliability of decentralized networks.

4.7.3 Elliptic-curve cryptography

Elliptic Curve Cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. This cryptography allows smaller keys compared to other types of cryptography to provide equivalent security [34]. ECC is the foundation of the use of private keys and digital signatures in Ethereum [3], and it can be also used to reduce the execution cost of the transactions, due to the use of smaller keys. For example, a 256-bits ECC key offers a security level of 128-bits, according to the NIST recommendation [35]. The better performance of ECC cryptography is also reviewed in [36], explaining their extensive use in blockchain technology.

At its core, ECC utilizes the properties of elliptic curves to provide secure encryption, digital signatures, and key exchange mechanisms. An elliptic curve is a mathematical curve defined by an equation of the form $y^2 = x^3 + ax + b$, where a and b are constants. The curve's points form a group structure, allowing for mathematical operations such as point addition and scalar multiplication.

The security of ECC is based on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP). Given a point P on the curve and a scalar k, determining the scalar k from the point kP is computationally challenging. This property forms the foundation of ECC's security.

ECC is widely used in various cryptographic protocols and applications. Here are some key aspects of ECC:

1. Key Generation: In ECC, a user generates a private key, which is a randomly selected scalar. The corresponding public key is obtained by scalar multiplication of a standard base point on the curve by the private key.

2. Key Exchange: ECC enables secure key exchange using protocols such as Elliptic Curve Diffie-Hellman (ECDH). Two parties, each with their private key and the other party's public key, can compute a shared secret using scalar multiplication.

3. Encryption: ECC can be used for symmetric key encryption through techniques like Elliptic Curve Integrated Encryption Scheme (ECIES). It combines the security of ECC with symmetric encryption algorithms, allowing for secure data transmission.

4. Digital Signatures: ECC-based digital signatures provide integrity and authenticity of digital data. The Elliptic Curve Digital Signature Algorithm (ECDSA) is commonly used for signing and verifying digital signatures in ECC.

5. Performance: ECC offers significant advantages in terms of performance compared to traditional cryptographic algorithms. It requires smaller key sizes while providing the same level of security. This leads to faster computations, lower memory requirements, and more efficient use of network bandwidth.

6. Security: ECC is considered secure against various cryptographic attacks when implemented correctly. The security strength depends on the size of the elliptic curve used and the selected cryptographic parameters.

In [37] there is defined the set of elliptic curves recommended for the U.S. Government use, and different alternative representations for these curves to allow more implementation flexibility. With that, we can see the cryptographic parameters that must be published to ensure privacy and security according to international standards.

The encryption systems for the purpose of data confidentiality are defined in [38], and it can be applied to ECC, like in [39], where it's provided an extensive review of the Elliptic Curve Integrated Encryption Scheme (ECIES), the best-known scheme based on ECC, to be used as a tool for encrypting data, creating digital signatures or performing key exchanges.

With all of that, we can conclude that ECC cryptography is the better alternative to improve security and preserve privacy in the data stored in the blockchain. There are several studies like [40] and [41] that have already applied this cryptography to blockchain technology to achieve these results without prejudice to the cost of transactions.

4.7.4 Identity-based encryption

Identity-Based Encryption (IBE) is a form of public-key cryptography that simplifies traditional key management challenges by allowing a user's public key to be directly derived from identifiable information, such as an email address or a username. This approach, proposed to alleviate the complexities associated with the Public Key Infrastructure (PKI), marks a significant shift towards a more manageable cryptographic framework.

At the heart of IBE is the concept of a Private Key Generator (PKG), which uses a master secret to generate private keys from users' public identifiers. This process typically relies on advanced mathematical constructs, such as bilinear pairings on elliptic curves, to secure the encryption scheme against potential attacks.

Identity-Based Encryption has been already used in different solutions, like making high-security multicasting in wireless sensor networks [42], to enhance authentication among the various communicating devices within wireless sensor networks during broadcasting for secure end-users message distribution [43], or to overcome the key management issue but still guarantee security even when attackers corrupt the keys, extending the FS-PEKS scheme (Lattice-Based Forward Secure Public-Key Encryption with Keyword Search) [44].

Despite its advantages, IBE is not without its challenges. The key escrow problem remains a significant concern, as the PKG's ability to generate private keys for any public identifier means it can decrypt all messages. Additionally, the computational demands of certain IBE schemes may render them impractical for resource-constrained environments.

Recent developments in the field of IBE aim to mitigate these limitations, focusing on enhancing the efficiency and security of IBE schemes. Innovations such as lattice-based cryptography offer promising routes for constructing more secure and efficient IBE systems.

In conclusion, Identity-Based Encryption represents a pivotal advancement in cryptographic technology, addressing key distribution challenges inherent to traditional PKG. As research continues to evolve, IBE stands on the cusp of broader adoption, promising to play a crucial role in the secure communication paradigms of the future.

4.7.5 Zero-Knowledge Proofs

Another cryptographic method that can be applied to blockchain technology to achieve certain properties is the *Zero-Knowledge Proof (ZKP)*. With this method, one party, the prover, can prove to another party, the verifier, that a given statement is true while the prover avoids sending any additional information apart from the fact that the statement is indeed true [45]. In other words, ZKPs allow one party to demonstrate knowledge of a fact or secret without disclosing any details about the knowledge itself. Since all information stored in the blockchain ledger is public, and all transactions are broadcasted to all participants, with ZKP we can allow users to share confidential information with security.

There are two main types of *Zero-Knowledge Proofs (ZKPs)*, interactive and non-interactive. In an interactive ZKP there are a series of tasks that the provers must complete to verify that they have particular information. This usually involves concepts of mathematical probability. In a non-interactive ZKP, there is no need for interaction between the prover and the verifier. Typically, the validation of the proof relies on computational assumptions. One example of a *non-interactive ZKP* is the *Schnorr Non-Interactive Zero-Knowledge (NIZK) proof* [46]. With the *Schnorr NIZK proof* anyone can prove the knowledge of a discrete logarithm without leaking any information about its value.

ZKP has the potential to encrypt data in pieces, enabling users to control certain blocks and the visibility of the information contained within them, allowing some users access while restricting others.

There are some surveys about the application of *Zero-Knowledge Proof* to the blockchain technology, like [47] and [48] that identify some potential problems of the ZKP application and future research directions and show the application of ZKP in usage in blockchain in real life examples. [49] follows an approach based on Zero-Knowledge Proofs as a verifiable computation technique to prove the correctness of a differentially private query output.

ZKPs can also be used to enable secure and private authentication and verification of identity. For example, an individual could use a ZKP to prove their identity to a third party, such as a bank or government agency, without actually revealing any personally identifying information. This can help to reduce the risk of identity theft and other types of fraud, while still enabling individuals to maintain control over their personal data.

Overall, Zero-Knowledge Proofs have the potential to greatly enhance the security and privacy of Self-Sovereign Identity systems, by enabling individuals to selectively share only the specific pieces of information that are needed for a particular transaction or interaction, while keeping the rest of their personal data private and under their own control.

zkSNARKs

A Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARK) is a specific type of Zero-Knowledge Proof that allows for very efficient verification of complex computations. zkSNARKs are especially useful for blockchain applications, particularly when privacy and scalability are crucial factors [50] Its key advantage is that they can be used when there is no possibility that the prover and the verifier could interact in real-time. They are used in blockchains like Zcash [51], where they enable users to prove that they have the right to spend a certain amount of cryptocurrency without revealing their account balance or the identities of the sender and receiver. In [52] we can read a formal and complete model of zkSNARK.

Circom[1] [53] is an open-source toolkit for creating zkSNARK circuits, which is designed to be more user-friendly and efficient than other circuit compilers. It allows developers to write circuits in a high-level language, which is then compiled into an optimized arithmetic circuit that can be used to generate zkSNARK proofs. Circom also provides a tool called *snarkjs*, which can be used to create and verify proofs. Further details about these tools can be found in [54].

In combination with Circom, it is highly recommended to use Poseidon as a hashing algorithm [55]. Poseidon was specifically designed for use in zkSNARK, and is already used in some blockchain-based applications on Ethereum.

In summary, zkSNARKs are a powerful cryptographic tool that enable efficient verification of complex computations without revealing any additional information, and Circom is a popular toolkit for developing zkSNARK applications.

Schnorr Non-Interactive Zero-Knowledge Proofs

Schnorr Non-Interactive Zero-Knowledge (NIZK) proofs [46] offer a distinctive approach to achieving zero-knowledge without interaction between the prover and the verifier. Leveraging the simplicity and efficiency inherent in Schnorr signatures, Schnorr NIZKs are constructed through a well-known cryptographic technique called the Fiat-Shamir transformation. This transformation converts an interactive proof system into a non-interactive one by employing a cryptographic hash function to simulate the challenges that would typically be provided by the verifier in an interactive setting.

The appeal of Schnorr NIZKs lies in their concise proof size and the minimal computational overhead, making them particularly suitable for applications where bandwidth and computational resources are limited. Unlike zkSNARKs, Schnorr NIZKs do not require a trusted setup, thereby mitigating certain security risks associated with the setup phase. This characteristic makes them advantageous for blockchain applications that prioritize transparency and security.

Schnorr NIZKs are highly regarded for their robustness and efficacy in cryptographic applications. Their strength is particularly advantageous for blockchain protocols, which demand high levels of integrity and security for cryptographic operations. These proofs are crucial in environments like blockchain-based e-commerce protocols, where it is essential to verify the authenticity of a transaction without revealing its contents. This ensures that transaction integrity is maintained while preserving the privacy of the transaction details.

[1] https://docs.circom.io/

In this thesis, Schnorr NIZKs are utilized in various blockchain e-commerce protocols to reinforce security without sacrificing efficiency. Their integration into these systems is crucial, enhancing the cryptographic toolkit available to blockchain developers and contributing significantly to the robust security architecture of these protocols.

4.8 Design patterns for smart contracts

Smart contracts are self-executing contracts with the terms of the agreement directly written into code. As blockchain technology, particularly Ethereum, has evolved to support complex applications, the need for efficient and secure design patterns in smart contract development has become paramount. This section examines various design patterns that enhance the functionality, security, and efficiency of smart contracts. We explore foundational patterns such as Factory smart contracts, and address advanced implementations like the Minimal Proxy Contract (EIP-1167), which optimizes deployment costs and contract interactions. By integrating these design patterns, developers can build more robust, scalable, and secure decentralized applications, pushing the boundaries of what blockchain technologies can achieve in various sectors.

4.8.1 Factory smart contracts

The concept of factory smart contracts plays a significant role in the management and deployment of new smart contracts within a blockchain ecosystem. A factory smart contract, also known as a contract factory, serves as a template or blueprint for creating and initializing new instances of smart contracts.

The primary purpose of a factory smart contract is to streamline and automate the process of deploying new contracts. It provides a standardized approach for creating new contracts with predefined rules, parameters, and functionalities. The factory contract typically contains the logic and code necessary to create and manage new instances of smart contracts.

Here's a simplified overview of how a factory smart contract operates:

1. Contract Deployment: The factory smart contract is deployed on the blockchain, becoming a permanent and immutable part of the network.

2. Contract Creation: When a user or entity wants to create a new smart contract, they interact with the factory contract and provide the necessary input parameters. These parameters might include contract-specific details, such as contract name, addresses of involved parties, initial settings, or any other required information.

3. Initialization: The factory contract takes the input parameters provided by the user and uses them to create a new instance of the smart contract. The initialization process involves executing the necessary code to set up the new contract with the desired configurations.

4. Contract Registration: Once the new contract is created, the factory contract may also take on the responsibility of registering and keeping track of the newly

deployed smart contracts. It maintains a record or registry of all the contracts created by storing their addresses or other identifying information.

5. Management and Upgrades: The factory contract can provide additional functionalities to manage the registered contracts. It can include features like contract ownership transfer, access control mechanisms, upgradeability options, or other administrative capabilities. These management features ensure that the deployed contracts can be effectively controlled and modified if needed.

By utilizing a factory smart contract, the process of creating and managing new contracts becomes more efficient and standardized. It reduces the need for manual intervention and enhances the overall reliability and transparency of the contract deployment process. Moreover, the factory contract can be designed to enforce specific rules or conditions for creating new contracts, ensuring compliance and consistency across the network.

It's important to note that the exact implementation and features of factory smart contracts may vary depending on the blockchain platform or programming language used. Developers have the flexibility to design the factory contract according to their specific requirements and the capabilities of the underlying blockchain technology.

4.8.2 Factory clone smart contracts and EIP-1167: Minimal Proxy Contract

The Factory Clone programming pattern is a better gas-cost solution than the traditional Factory method programming pattern, achieving an easier management system for the newly deployed smart contracts and also a simple storage method for the addresses.

The *clones*, also known as *minimal proxies*, work on the same objective as the traditional factory but also reduce the elevated gas costs. It is explained in the standard ERC-1167 [56], which provides a very simple functionality to clone a contract in an immutable way. To achieve this functionality, the standard presents a minimal Solidity bytecode implementation that delegates all calls to a known and fixed address.

Using the implementation with clones, instead of deploying a smart contract, we can deploy a cheaper minimal smart contract that points to a smart contract template that has previously been deployed on-chain. So, the minimal contracts delegate all calls to the implementation of this main smart contract that is used as a reference.

Here's how it generally works:

1. A master contract, containing all the necessary logic, is deployed once.

2. The factory contract, which holds the address of the master contract, can create new clones. These clones are lightweight and contain almost no logic.

3. When a function call is made to a clone, it delegates the call to the master contract using Ethereum's *delegatecall* feature. This means the master contract's code is executed in the context of the clone's storage, allowing each clone to behave as if it were a fully independent contract.

The benefits of using EIP-1167 include:

1. **Reduced Deployment Costs**: Since clones are minimal and share the master contract's code, deploying them is much cheaper than deploying full contracts.
2. **Upgradeability**: If the logic in the master contract needs to be updated, only the master contract needs to be redeployed and updated in the factory. All clones can remain untouched but will use the new logic due to *delegatecall*.
3. **Consistency and Reusability**: Using a standard pattern for clones ensures consistency across different projects and allows developers to reuse existing, well-tested implementations.

However, it's important to note that while EIP-1167 and factory clones can provide significant gas savings and flexibility, they also introduce complexity and potential security risks, particularly around the delegation of calls and shared logic. It's essential to thoroughly understand and test the implications of using such patterns in our smart contracts.

4.9 Blockchain-related infrastructures

Beyond token standards and cryptographic technologies, a range of infrastructural technologies significantly enhance blockchain-based e-commerce protocols. Among these, the InterPlanetary File System (IPFS) stands out as a key component for decentralized and efficient storage solutions. As a Peer-to-Peer distributed file system, IPFS utilizes content-addressable storage to improve the availability, resilience, and censorship resistance of digital assets. This technology enables the elimination of centralized servers, providing a robust infrastructure for hosting product images, descriptions, and other vital e-commerce media. Furthermore, IPFS optimizes the retrieval and distribution of files through a content-based addressing system, which greatly enhances the scalability and accessibility of e-commerce platforms.

4.9.1 InterPlanetary File System

InterPlanetary File System (IPFS) provides a distributed Peer-to-Peer system where users can store any type of content, that will be shared with all the network users, getting a censorship-resistant system. The main difference with the HTTP protocol is that IPFS uses content addressing, unlike HTTP protocol which uses location addressing.

The IPFS content addressing protocol, identifies the content by a Content Identifier (CID), a cryptographic hash of the content that is used as the content address. The users send it to the IPFS nodes to obtain the content. This protocol provides immutability of the stored content, for the simple fact that if stored data is edited then its CID will change [57].

In the implementation of the protocol described in Chapter 11, we have used the Infura IPFS gateway to perform the notifications upload and download. The Infura IPFS service also provides us the pinning of the documents until six months after the last access, or the document upload. Also, an own IPFS node could be used, then there wouldn't be any type of time limit for pinned files.

The pinning service fixes permanently a CID content to a node so this content is excluded from being deleted by the garbage collector system [57] that periodically

removes the not recently used content stored in a node, in order to clean part of the node memory [58]. In other words, the pinning service prevents certain content from being deleted completely from the network.

CHAPTER 5

USE-CASES OF THE BLOCKCHAIN TECHNOLOGY

The primary use of blockchain technology was as a distributed ledger for cryptocurrencies such as Bitcoin, but as an unalterable database, it can enable new solutions for a wide range of traditional applications: financial services, games, trading, healthcare [59, 60, 61, 62], etc... Also, blockchain technology has been used in Internet of Things (IoT) systems; for instance, [63, 64] propose schemes where blockchain-based IoT has the potential to create more feasible solutions than cloud-based IoT systems in terms of consumption, maintenance, and upgrade costs, and more effective in terms of access control and authentication management. Moreover, blockchain-based proposals expand their scope in many areas in combination with IoT such as transportation systems. [65] highlights the factors that affect the design of a trust model for the so-called social Internet of vehicles and explains how blockchain technology can be used to preserve privacy in such systems. In [66], blockchain technology is applied to the supply chain and logistics methods to improve product visibility, tracking, and process automation. The most prominent fields of applications of blockchain technology are summarized in [67]: Financial Applications, Smart Contract as law intermediaries, Blockchain in combination with IoT, Securing Communications, Medical Data, and Blockchain as a decentralized platform for Artificial Intelligence. Thus, blockchain technology, as an emerging and disruptive technology, is proposed to enhance the security of different kinds of applications that range from voting systems or the aviation industry to energy management or mobile payments [68, 69, 70, 71] in order to provide trusted transparency, immutability, traceability, and auditability for the exchanged and stored information.

Another potential application of blockchain is in electronic tickets (e-tickets), which represent a user's right to access services and can be transferred to others along with their associated rights. Various research works focus on achieving security, privacy, and functionality in e-ticket systems, addressing aspects like user exculpability [72], reusability [73], and transferability [74, 75]. The advent of blockchain and Non-Fungible

Tokens (NFTs) introduces a novel framework for redefining e-tickets. NFTs, primarily hosted on blockchain networks like Ethereum, encapsulate ownership and are based on standards like Ethereum's ERC-721, ensuring the authenticity of assets ranging from artworks to tickets [76]. This blockchain-based innovation meets the stringent security and privacy demands of e-tickets [77, 78], encouraging opportunities in various applications including transportation, where the integrity and confidentiality of tickets are capital.

In recent years, there has been a significant surge in interest and adoption of blockchain-based finances, commonly referred to as *Decentralized Finance (DeFi)*. This financial paradigm eliminates the need for traditional central financial intermediaries like banks or exchanges to facilitate various financial services such as lending, trading, earning, borrowing, or staking. Instead, these services are provided through the utilization of smart contracts, which are self-executing agreements coded on blockchain platforms like Ethereum.

A notable milestone in the growth of DeFi occurred in October 2021 when the total value of assets utilized in decentralized finance surpassed $100 billion for the first time. This data was compiled and reported by *DeFi Pulse* [79], a leading platform that tracks and analyses the progress of the DeFi ecosystem.

However, this thesis aims to explore a different aspect of blockchain technology, specifically focusing on its applications within the e-commerce sector. This research will delve into various use cases that leverage blockchain, including certified notifications or registered eDeliveries, contract signing, micropayments, and digital identity.

By examining these specific use cases of blockchain technology within the e-commerce domain, this thesis seeks to shed light on the potential benefits, challenges, and implications of integrating blockchain into existing e-commerce infrastructures.

5.1 Certified notifications or registered eDeliveries

Certified notifications, or **registered eDeliveries**, include an exchange of elements between the sender and the receiver or set of receivers. In these kinds of deliveries, the sender sends a message, and then the receiver can read it, sending proof of reception to the sender. These notifications don't have a definitive and standardized solution in their electronic version. For example, a notification can be done using electronic mail, and there are several proposals for this service. However, not all certificated notifications require the use of electronic mail. The term "*registered eDeliveries*" is employed by the Regulation (EU)910/2014 of the European Union Agency for Network and Information Security [80]. This will be further detailed in subsection 5.1.1.

Almost all the existing proposals for certified notifications include the existence of a *Trusted Third Party (TTP)* to solve disputes and to ensure fairness between the different parties of the exchange. This TTP can play an important role (pessimistic protocols), participating in all the steps of the exchange, or a more relaxed role (optimistic protocols), where TTPs are only active in case of dispute between the parties.

Fair exchange protocols proposed so far usually use TTPs [81, 82, 83], which are responsible for resolving any conflict that arises as a result of interrupted exchanges or fraud attempts. In addition to that, these protocols normally use non-repudiation mechanisms in order to generate evidence that proves the behavior of the actors of

the protocol. Currently, with the advent of blockchain technology and smart contracts, TTPs can be replaced or complemented by this new know-how, which opens a range of new possibilities to find effective solutions to the electronic versions of the protocols that fulfill the generic pattern of fair exchange of values. A method for designing new fair exchanges by means of the Bitcoin network is to motivate parties to complete the protocol in order to ensure fairness by using a bond or a monetary penalty for dishonest parties [84, 85, 86].

The ideal properties for fair certified notification protocols [87, 88, 89] are:

- **Effectiveness**. If both parties have the correct behavior, they will receive the expected items.
- **Fairness**. After completion of the phases of the protocol, either each party receives the expected item or neither party receives any useful information about the other's item. We can have **weak fairness** if at the end of the execution, both parties have received the expected items or if one entity receives the expected item and another entity does not, the latter can get evidence of this situation. A multiparty protocol is said to be **fair** if, at the end of the protocol, all parties receive what they expect or none of them receive any valuable information.
- **Timeliness**. At any time, each party can unilaterally choose to terminate the protocol without losing fairness. A multiparty protocol is said to respect timeliness if all honest entities can terminate the protocol in a finite amount of time without losing fairness.
- **Non-repudiation**. If an item has been sent from a sender to a receiver, the sender cannot deny the origin of the item and the receiver cannot deny receipt of the item.
- **Verifiability of Third Party**. If the third party does not behave as expected, resulting in the loss of fairness for a party, any other party can prove this fact in a dispute.
- **Confidentiality**. Only the sender and the receiver of the notification know the contents of the certified message. A multiparty protocol is said to be confidential if only the aimed honest recipients can reveal the message.
- **Efficiency**. An efficient protocol uses the minimum number of steps to allow an effective exchange at a minimum cost.
- **Transferability of evidence**. The proofs generated by the system can be transferred to external entities to prove the result of the exchange.
- **State Storage**. If the TTP is not required to maintain state information when it needs to be involved in the exchange, then the system is stateless.

Some of the above properties cannot be achieved in the same protocol. The authors of [90] enumerate the incompatibilities among the ideal features. Some examples are Weak Fairness and Transferability of Evidence, Stateless TTP and timeliness, and Verifiability and transparency of the TTP. In this research, we will see that in a protocol

that offers weak fairness, a party cannot transfer the evidence to an arbiter since the other party could have contradictory evidence.

There can exist incompatibility among some of the properties and it can be difficult to achieve simultaneously some other properties. Some protocols solve the exchange efficiently with an optimistic TTP but only achieve weak fairness [91]. Other systems focus on the achievement of specific features such as the transferability of the evidence [92] the verifiability of the TTP [93], the avoidance of the selective rejection based on the identity of the sender [94], the flexibility to allow the delivery to multiple receivers [95] or the reduction of the volume of state information that the TTP must store [96].

There are surveys like [88] that analyze the properties of these exchanges in multi-party scenarios. In the same way, as in the exchanges between two parties, the literature does not agree on the properties that the protocols must offer, which is why a compromise is made between properties, depending on the specific application.

There is a dominant trend in the use of an optimistic approach of the *Trusted Third Party* in multiparty protocols. For example, in [97] there is a protocol that uses group signatures with an online TTP, [95] presents an efficient multiparty optimistic protocol with the properties of asynchrony and verifiability of the TTP while [98] describes a more efficient, asynchronous optimistic protocol.

There are some previous studies [99, 100, 101, 102] on fairness using blockchain that focus on fair purchase operations between a product or a receipt in exchange for cryptocurrencies (usually Bitcoin).

In [103] we can see, for the first time, a smart contract for the resolution of a purchase operation while [104] uses smart contracts and trusted execution environments to guarantee the fair exchange of payment and the result of execution. The authors investigate the fair exchange problem and propose two solutions for fair payment using smart contracts supported by the functionality of blockchain with a Turing-complete language. Thus, most of the solutions using blockchain technology for fair exchanges are focused on performing atomic payments. For example, in [105] Delgado-Segura et al. propose a protocol based on Bitcoin scripting language for fair exchange between data and a payment where the seller of the data cannot cash in the payment if the buyer has not obtained the data and the buyer will not get the data without executing the payment. In [85] some variants of a protocol for exchanging Bitcoins for digital goods are presented by using an escrow system. Also in [106] another problem related to fair exchange and blockchain technology is studied. The authors propose a solution for a Proof-of-Delivery of physical assets without the involvement of any TTP. The provided solution can be applied to many shipped physical items in exchange for a payment.

In [107], Hasan et al. propose a non-repudiation protocol using blockchain technology and the Ethereum smart contracts. This solution is applied to a trade exchange between crypto-tokens and digital assets. The protocol requires a deposit of collateral by the involved parties to incentivize the honest behavior of the actors. Thus, this protocol is only appropriate for trade scenarios to provide proof of delivery but the fairness of the exchange between sender and receiver is not proven. Esposito et al. in [108] describe a possible blockchain notification system for mobile apps. The system is applied to event-based subscriptions supported by a blockchain infrastructure deployed as a cloud service, however, the concision of the proposed system is not clear and no smart contracts are proposed to secure the messaging scheme. Also, Zupan et al. in [109] propose a notification system based on smart contracts deployed on a

blockchain using Hyperledger Fabric. Although the system provides authentication via certificates issued by Fabric's CA, this scheme does not provide either a fair exchange scheme of delivery notifications or proof of reception of the issued information.

A multiparty certified notification allows the sender to send a message efficiently to multiple receivers. Typically, such protocols employ TTPs in optimistic approaches to manage the exchange and do not utilize blockchain technology. For instance, [110] presents a proposal for a multiparty fair exchange that operates optimally for any network topology, requiring only a constant number of rounds and relying entirely on the TTP for ensuring fairness. Additionally, [88] describes a multiparty non-repudiation scheme as a protocol where *"n | n > 2 entities agree to use a non-repudiation protocol for exchanging messages (general or specific purpose) and collecting evidence of the transactions performed for the exchange of those messages"*. The method of message exchange can vary based on the application, with configurations such as ring or mesh networks or a one-to-many distribution approach. Despite these developments, there are currently no proposals that leverage blockchain to address multiparty fair exchanges.

5.1.1 Qualified electronic registered delivery services

In July 2016, Regulation (EU)910/2014 of the European Union Agency for Network and Information Security [80], also known as eIDAS Regulation, became applicable. This regulation establishes the rules on electronic identification and trust services for electronic transactions in the internal market and covers all 28 member states.

Several trust services are defined in the document: electronic signatures, seals, timestamps, registered delivered services and certificates for website authentication. The regulation introduces the notions of qualified trust service and qualified trust service providers with the requirements and obligations that ensure the high-level security of the trust service. Also, there are a series of documents, like [111] that aim to assist parties wishing to use the aforementioned services.

In the US we can find the FICAM Trust Framework Solutions (TFS)[1], the federated identity framework for the U.S. federal government. It includes guidance, processes and supporting infrastructure to enable secure and streamlined citizen and business-facing online service delivery. However, the framework does not include the specifications for the registered electronic delivery of data.

Certified notifications or registered delivery services are mainly offered by postal services in many countries and they have different denominations depending on each service provider. For instance, most postal services offer a Registered mail service that includes the sending of *lettermail*, documents and valuables. In the case of the United States Postal Service (USPS), the service is called Certified Mail, but it also offers a Registered Mail service that additionally provides the Chain of custody properties. In other words, the collection of information that provides evidence about the chronological actions in the delivery service or sequence of custody, control, transfer, analysis, and disposition of the delivery. Also, the Registered Mail service of the USPS can specify the delivery status or attempted delivery status when the item reaches its destination [2]. This way, these kinds of services provide evidence that a user who acts as a receiver (or set of receivers) has access to the data since a specific instant. As trust services, these

[1] https://www.idmanagement.gov/trust-services/

[2] (https://faq.usps.com/s/article/What-is-Registered-Mail#international)

proposals must offer a high level of security and protection of the privacy of the users but they also have to consider the regulations on the subject. Therefore, distributed ledger, data protection and immutability features of the blockchain technologies make the blockchain an ideal tool to offer trust and data trail for new certified notifications or eDelivery solutions.

We will focus on the electronic registered delivery service, also called eDelivery. For simplicity, we will use the term *Certified Notification* in the proposed protocols. Data sent and received using this service achieve the properties of the integrity of the data, the identification of the sender of the data and also of its receiver. Examples of the use of qualified electronic registered delivery services are electronic registered mail, official submissions in e-government services, access and exchange of sensitive data and notarization of events. It is important to highlight that the eIDAS regulation establishes the principle that an electronic document should not be denied legal effect because it is in electronic form.

Article 3.36 of the regulation [80] states that an eDelivery service is *"a service that makes it possible to transmit data between third parties by electronic means and provides evidence related to the handling of the transmitted data, including proof of sending and receiving data, and that protects transmitted data against the risk of loss, theft, damage or any unauthorized alterations"*.

A qualified eDelivery service must offer the following functionalities according to the directive: data integrity, authentication of origin (both natural person or legal person) and authentication of the time of the delivery. Confidentiality is not considered a core functionality but it is usually provided as part of a more complete solution. This consideration will be taken into account in the design of the protocols. For an eDelivery service in order to be a Qualified Electronic Registered Delivery Service according to [80], the service must be provided by qualified trust service providers whose compliance with the requirements is regularly checked by an accredited entity.

The data, that can be sent in an eDelivery service from a sender to a receiver, can be of any type (thus, it includes electronic documents) and the transmission means can be of any kind. Usually, the data referred to by the definition of eDelivery services is generally called a "message". This way, certified notifications and certified electronic mail are included in the e-Deliviery services. Email is a means of transmission that can be used, but eDelivery is not limited to email. Registered email is a general-purpose implementation of eDelivery [111], whereas there are more specific eDelivery services.

Some use cases for eDelivery services are included in [111]:

- Electronic registered mail, an enhanced form of email transmitted by electronic means that provides evidence relating to the handling of an e-mail including proof of submission and delivery.
- Delivery of official electronic notifications and supporting official submissions in eGovernment services.
- Accessing and exchanging sensitive electronic data.
- Electronic notarization of events.

5.2 Contract signing

Contract signing is increasingly used in electronic commerce and digital government to implement electronic signatures in a cryptographically protected way. An electronic signature is a juridic concept, where a person validates the content of an electronic message through any electronic medium that is legitimate and permitted. One of these electronic mediums is a digital signature, a cryptographic mechanism that lets anyone verify the authenticity and integrity of a message, checking that a message was created by a known sender and that the message was not altered in transit.

This electronic signature of a contract can't be done simultaneously and in the same place by both parties. For this reason, we need a fair exchange protocol that can also follow the regulations for this kind of service, such as [80] or [112]. In these documents, we can see that there exist some legal concerns like the proofs generated using the signature service and the time of the signature of the contract. These proofs will be useful in case of disputes among the signers. For example, in [112] it is described a model that has three steps: *offer, acceptance, and acknowledgment* of the acceptance.

Typically, the electronic contract signing depends on a *Trusted Third Party (TTP)* that can manage the exchange and solve the conflicts that could arise among the signers. There are protocols, like [113, 82, 114, 115], that involve more or less these TTPs, from **in-line protocols** where the TTP is involved in all the steps, **on-line protocols** where the TTP is involved only in certain steps of the signature, and **optimistic protocols** where the TTP is only involved in case of disputes among the signers or in case of execution problems. The protocols that need the presence of a TTP have problems like low performance, high costs, and security concerns. With that, removing the TTP from the protocol would be very useful and efficient for electronic contract signing protocols, like in [116].

There are also different proposals for multiparty contract signing protocols. One of the biggest issues in the multiparty signature is efficiency. For example, in [115] we can see a low performance in the solutions, because when the number of signers increases, the execution costs, the number of interactions among the parties, and the general complexity of the protocols also increase.

Blockchain can be included in the design of contract signing protocols, but like in other protocols, the data and state changes that involve the different participants of the signature will be stored in the blockchain ledger. On the other hand, the time when the transactions are performed is also stored in the blockchain. With that, we will have all generated evidence of that signature in the blockchain ledger, but this is to the detriment of privacy properties, such as confidentiality of data.

There are several proposals to manage the exchange of signatures in contract signing using blockchain technology. For example, [117] proposes a system that stores signed documents in the blockchain using RSA signatures embedded in a PDF file. [118] proposes a complex solution for the signature of a contract that is limited to three signers, suitable for the fog computing paradigm. This is a very expensive proposal because it uses the threshold public key encryption and verification cryptography to provide fairness to the exchange, which is computationally very expensive. In [119] we can see a solution that uses the Emercoin blockchain and applies its Name-Value Storage (NVS) technology to propose a method for multilateral data signing. In this solution, the authors claim that "presumably" the solution could be useful to sign contracts and

deeds. But this contract signing scheme can be only applied with the NVS technology on the Emercoin blockchain. The solution proposed in [116] presents a contract signing protocol with proof of existence that also uses blockchain technology. It's based on a three-step pattern, and the blockchain ledger is not used to store the contract, it's used to timestamp a hash of the contract. However, this protocol is not able to check the integrity of the contract to be signed. [120] presents another solution that uses the Bitcoin blockchain and smart contracts to create a multiparty fair exchange in a star topology that could be applied to the exchange of signatures. This protocol guarantees fairness, but it does not consider some properties of contract signing protocols and it is not as efficient as an explicit contract signing system.

5.3 Micropayments

E-commerce evolves day by day, introducing new applications and services that transform how we interact with digital markets. Among these innovations, **micropayments** facilitate the payment of small amounts for low-cost goods or services, a process commonly referred to as **micropurchases**. These transactions attend to unique security and functional requirements within the field of electronic payments. Primarily utilized for acquiring intangible goods such as digital media (music, videos), electronic data, virtual gifts, and access to premium digital content (e.g., newspapers and product reviews), micropurchases involve low-value transactions that require highly efficient payment systems to remain cost-effective.

The principal challenge in facilitating micropurchases lies in minimizing operating costs—including transaction fees and processing overheads—to ensure profitability for both sellers and buyers. This is crucial in services like location-based services where transactions are frequent but involve small amounts. By simplifying the payment process and reducing associated costs, micropurchases enable users to economically acquire digital content and services on a per-use or subscription basis. This streamlined financial mechanism is essential for the monetization strategies of companies dealing with digital content and services, ensuring that the dynamic nature of e-commerce is supported by equally dynamic and efficient payment solutions.

First, we need to consider that security properties are a primary concern for the development of micropayment systems to avoid financial risks for merchants and also to ensure customer privacy. On the other hand, the efficiency and cost of individual transactions are critical factors for the development of these systems. However, efficiency and security are generally opposed, so micropayments must provide a trade-off between these requirements.

Cryptocurrencies, digital means of exchange that we have talked about in previous sections, have many advantages over fiat currencies, but they also have limitations, such as the high costs of transactions. In addition, the operations carried out on the blockchain are not completely anonymous, since they can be traced by being publicly visible. But users cannot be easily identified, as they use addresses that are pseudonyms.

We can make instant transactions outside the blockchain using so-called **payment channels**, which will allow us to make instant transactions with cryptocurrencies. These allow us to overcome the problems of the blockchain in terms of high costs and scalability, in terms of transactions per second and the space occupied in it. The payment

channels are based on the inclusion of many operations in the same transaction of the main blockchain. Therefore, they are an excellent second-layer solution that eliminates direct dependency on the blockchain.

[121] presents an efficient and secure micropayment scheme, where a fair exchange between the currency and the desired good or service is described, ensuring anonymity and the impossibility of tracing customers. Rivest et al. [122] introduce two simple micropayment schemes, *PayWord* and *MicroMint*, which use offline hash chains, but it still didn't use blockchain technology.

The emergence of the Ethereum blockchain has led to the proposal of a multitude of solutions with micropayment channels. For example, these kinds of channels on the Ethereum network have already been discussed in [123], trying to improve their performance and cost. The protocol proposed in this article scales logarithmically with the capacity of the channel, using a variant of the *Merkle tree*, and does not require the payer to lock the entire balance at the creation of the channel. In [124] Di Ferrante also showed how to build payment channels in Ethereum using only *"50 lines of code"*, between the payer and the payee. The smart contract implemented allows verifying the digital signatures of payments made outside the blockchain using the operation code *ecrecover*, which returns the address of the signer. [125] also uses Ethereum technology, proposing the decentralized Hybrid cryptocurrency Exchange (HEX) to combine the benefits of CEX (centralized) and DEX (decentralized) token exchanges. This HEX extends existing solutions by adding a new payment channel layer to benefit frequent merchants and alleviate backlog congestion.

In [126] was proposed the use of a network of *multihop* and anonymous payment channels. The solution is based on *Elliptic Curve Cryptography (ECC)* and has been proven to be secure while achieving payment path privacy and sender and receiver anonymity.

There are also other networks such as **Raiden** [127], a network for instant transactions, which operates on the Ethereum platform, that aims to solve the scalability problem that currently has this network. *Raiden* is extended in [128] to implement the *payGo* protocol. The results indicate that, in comparison with the conventional linear pricing for latency, *payGo* can improve about 90% of the payer's utility by optimizing latency and incentive. *Raiden* network is an analog to the idea of the **Lightning Network** [129], a second layer payment protocol designed to be layered on top of blockchain-based cryptocurrencies such as Bitcoin. It is intended to enable fast transactions among participating nodes and proposes a solution to the Bitcoin scalability problem.

In the case of cryptocurrencies, the solution to implement micropayments lies in designing layer two protocols [130] in which each payment operation is prevented from representing a transaction on the blockchain. A channel, once opened, allows off-chain operations to be carried out that will not materialize on the blockchain until the channel is settled.

However, although payment channel networks have tried to mitigate the scalability problems inherent to blockchain networks, they still do not provide significant security and privacy guarantees, as demonstrated in [131].

5.4 Identity-related attribute verification preserving privacy

Self-Sovereign Identity (SSI) and Privacy-Preserving Identity (PPI) concepts are becoming increasingly prominent in the digital world, driven by growing concerns about data privacy and security [132, 133, 134]. Both are related concepts but have some key differences. Privacy-Preserving Identity focuses on maintaining user privacy and security during the identity verification process. It can involve the use of technologies such as ZKPs to verify user identities without revealing any personally identifiable information. Self-Sovereign Identity, on the other hand, is a broader concept that includes giving individuals complete control over their personal data and identities [135]. It is based on the principle that individuals should be the owners and controllers of their own identity information, rather than relying on centralized authorities or intermediaries.

While both Privacy-Preserving Identity and Self-Sovereign Identity aim to address issues around privacy and security in identity verification, they differ in their approach. Privacy-Preserving Identity focuses on the verification process itself, while Self-Sovereign Identity is a more comprehensive concept that encompasses the entire identity management system, including ownership and control of personal data. Windley's book, "Learning Digital Identity: Design, Deploy, and Manage Identity Architectures," provides further insights into how digital identity architectures can be designed and managed effectively, supporting both privacy-preserving techniques and Self-Sovereign Identity principles [136].

Decentralized digital identity management systems face several significant security challenges. A primary challenge is ensuring the confidentiality of personal data to prevent it from being publicly available. Identity and personal information theft are critical concerns, where attackers try to commit fraud using sensitive user data through malicious actions [137].

Current threat landscapes reveal that decentralized systems are vulnerable to various specific attack vectors. Including data breaches where unauthorized entities gain access to sensitive information, phishing attacks aimed to mislead users into revealing personal data, and social engineering attacks to exploit human behavior to obtain sensitive information.

Moreover, a digital identity management system must ensure the integrity and authenticity of the identity attributes. It should be possible for an authorized external party to verify the attributes, ensuring effectiveness and the transferability of the data evidence.

In addition to these common risks digital identity protocols must address additional challenges associated with the underlying technologies used in the implementation. A major concern is the usability of the protocol which in the usage of blockchain technology is highly related with the key management. Concern that can be mitigated by providing a secure key management system for users or by implementing an account abstraction system.

Furthermore, the selected blockchain network must address important risks and potential attacks. It should offer scalability to handle a large number of transactions and be secure against various threads, including Sybil, Denial of Service (DoS), Distributed Denial of Service (DDoS), Domain Name System (DNS) attacks, and 51% attacks [137]. The robustness of the network against these threats is critical to the overall security and effectiveness of the digital identity management system. For this reason in the

presented evaluation of properties we define a group of general assumptions mainly related to the selected blockchain, along with the assessment of the correct achievement of the above mentioned security properties.

Christopher Allen first introduced the term Self-Sovereign Identity (SSI) in 2016 [138], emphasizing the principles of freedom and decentralization within such systems. Mazzocca et al. provide in [139] a comprehensive survey of Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs) within SSI systems, marking a significant transition from centralized to decentralized digital identity management. It offers an overview of DIDs and VCs, standardized by the World Wide Web Community (W3C), to enhance secure and decentralized digital identity systems. The paper reviews available implementations and details their applications across various use-case scenarios beyond SSI systems, examines recent regulations and global initiatives related to these technologies, and identifies challenges hindering their real-world adoption while suggesting directions for future research. This extensive analysis aims to fill the gap in the existing literature by providing a thorough understanding of DIDs and VCs, their applications, regulatory landscape, and potential future developments. In their work, Satybaldy et al. [140] discuss the evolution of digital identity systems, positioning SSI as the subsequent stage in identity management. The discussion underscores the pivotal role of blockchain technology and distributed ledgers in advancing SSI systems. Conversely, Keršič et al. [141] leverage SSI principles to demonstrate the feasibility of voting in Decentralized Autonomous Organizations (DAOs). Their publication outlines the creation of SSI-based DIDs and the management of VCs, ultimately serving as voting tickets in a DAO. However, it's worth noting that their approach does not incorporate SBTs or ZKPs.

The user experiences significant benefits through the decentralization of digital identity management, attaining full ownership of their identity. Mecozzi et al. [142] provides a comprehensive overview of the current landscape of Identity and Access Management within the blockchain realm, along with relevant regulatory frameworks and guidelines.

The use of SBTs in SSI systems is explored in [143], while [144] provides further insights into the potential of Non-Transferable Non-Fungible Tokens for enhancing document traceability within decentralized identity systems. [145] introduces a novel OpenID Connect (OIDC) model leveraging Ethereum Blockchain and ERC-721 tokens to enhance security and privacy. Additional investigations, such as those conducted by [146, 147, 148], also delve into the utilization of SBTs. However, these studies exhibit significant limitations. Notably, none of them provide comprehensive details on the encryption and verification methods for the associated data. This lack of specificity hampers the understanding of how SBTs can ensure data security and privacy within SSI frameworks. Moreover, the absence of a unified approach to integrating ZKPs with SBTs to strengthen privacy remains a critical gap in the literature. As a result, these proposals fall short of demonstrating a robust and scalable solution for secure, privacy-preserving and non-transferable credential verification in decentralized identity systems.

Numerous existing proposals in the literature improve privacy in identity verification contexts through the utilization of ZKPs. For instance, [149] is built on pairings, short signatures, commitment and ZKP cryptographic schemes, and [49] adopts a ZKP-based approach as a verifiable computation technique to validate the accuracy of a differentially private query output. Other studies, like [150] and [151], outline privacy-

preserving identity schemes, utilizing a blockchain-stored Merkle tree to ensure the user data is not altered. Although these solutions incorporate ZKPs, they omit storing the data on-chain, the use of SBTs, and the identity holder's public/private key for data encryption. In a more intricate system, Wang et al. [152] describe a decentralized identity system generating ZKP proof during the verification and storing it in a smart contract. To preserve anonymity, user-account mappings employ linkable ring signatures. Despite its complexity, this solution, involving Soul and Seed accounts, does not leverage the public/private key of the end user for data encryption. Other solutions like zkLogin [153] also use ZKPs to authenticate transactions, ensuring that the link between a user's off-chain and on-chain identities is hidden. [154] describe a blockchain-based decentralized Identity Management System (IMS) referred to as BADIMAC. This model ensures secure and transparent identity verification through a decentralized consensus-based design. Zero-Knowledge Proofs verify claims without disclosing data, while documents are encrypted with the owner's public key, ensuring they can only be decrypted by the owner. Verification requests reveal only necessary information with user consent, and the results are securely encrypted and returned to the requester. The proposal presented in [155] delves into digital identity in the Web 3.0 era, introducing a new authentication scheme using verifiable credentials. The paper employs blockchain, smart contracts, and cryptographic concepts like Public Key Infrastructure (PKI) and ZKPs in its theoretical framework. The main findings show that the proposed scheme improves user control, security, and privacy in digital interactions. The broader implications highlight a move towards self-sovereignty in digital identity management and less dependence on centralized authorities for online transactions. Song et al. in [156] propose a novel approach that separates identity verification and credential issuance into two distinct roles, significantly mitigating the risk of identity information leakage, a common issue in traditional schemes where a single entity performs both tasks. It uses linkable ring signatures to map the user's physical identity to a digital identity. It also introduces advanced techniques like cryptographic commitments, ZKPs, randomized signatures, cryptographic accumulators, and AES encryption to support proactive and passive identity revocation while preserving privacy. The study presented in [157] provides a comprehensive review of the literature on integrating ZKP technology into blockchain to enhance privacy in identity sharing while maintaining transparency. Notably, none of these works that propose ZKP-based mechanisms to improve privacy in identity verification contexts [149, 49, 150, 151, 152, 153, 155, 156, 157] utilize SBTs to hold the user's digital identity.

The Sovrin Foundation [158] outlines a decentralized SSI framework, empowering individuals and organizations to autonomously manage their digital identities without dependence on a central authority. This framework strategically integrates ZKPs as a crucial element for preserving privacy and upholding data integrity. Notably, the Sovrin Foundation identity protocol does not incorporate SBTs. Its primary emphasis is on delivering a decentralized SSI framework, with a particular focus on the overall architecture rather than the implementation of specific token models. There are other projects such as uPort [159], Holonym [160], and Iden3 [161] contributing to the development of SSI systems using blockchain technology. However, none of these projects integrate SBTs, public key cryptography, and ZKPs.

A privacy-preserving Multi-Factor Authentication (MFA) framework for zero trust networks is presented in [162]. It proposes a Distributed Authentication Mechanism

(DAM) that leverages the decentralized architecture of blockchain technology. The framework features a distributed One-Time Password (OTP) creation method, where each node generates a segment of a secret, which is then aggregated to form an OTP. A ZKP scheme is integrated to ensure privacy-preserving OTP verification. Additionally, modified ERC-721 tokens are used as unique, non-transferable authentication tokens. These tokens, which have a specific period of validity, are used for authenticating user access within their valid time frame. Therefore, while they provide a secure and controlled method for verifying user authentication, these tokens are not intended to hold or represent the user's digital identity.

CHAPTER 6

DEVELOPMENT TOOLS OF THE PROPOSED SOLUTIONS

In this thesis, several of the proposed protocols have been implemented as smart contracts on the Ethereum blockchain, developed using the Solidity programming language. This choice of technology leverages Ethereum's robust and versatile environment, allowing for the creation of complex, decentralized applications integral to e-commerce protocols. Following the development phase, each smart contract underwent testing through unit tests to ensure reliability and functionality. A critical aspect of our analysis focused on the gas costs associated with executing these contracts, providing insight into their efficiency and the economic feasibility of deploying them on Ethereum or other EVM-compatible networks. This evaluation is paramount, as gas costs directly impact the scalability and user adoption of blockchain-based applications.

In the literature, several resources, such as Prusty (2017) [163] and Antonopoulos (2019) [3], offer comprehensive methodologies and best practices for writing and testing smart contracts. These references typically advocate for the use of the *web3.js*[1] library for interacting with Ethereum nodes. However, diverging from this standard approach, our work employs *ethers.js*[2], a library known for its lightweight architecture and increasing popularity among developers. *ethers.js* offers a streamlined and efficient interface for interacting with the Ethereum blockchain, which has contributed significantly to its adoption and the robustness of our testing procedures.

For a subset of the proposed solutions, specifically the confidential and non-confidential certified notifications and the decentralized and confidential digital identity using ZKP, we have advanced to the development of user interfaces. These interfaces are designed to enhance user interaction, providing an intuitive and secure means for accessing the functionalities offered by our smart contracts. The development process of these interfaces, discussed in subsequent sections, adheres to principles of user-

[1] https://web3js.org/
[2] https://ethers.org/

centric design, ensuring that the end product is not only functional but also accessible to users with varying degrees of blockchain familiarity.

The creation and testing of these protocols, along with the strategic choice of development tools and libraries, underscore our commitment to developing scalable, efficient, and user-friendly blockchain-based e-commerce solutions. Our approach, detailed in the following sections, reflects a comprehensive understanding of both the technical and user-experience aspects critical to the successful implementation of these protocols in real-world scenarios.

6.1 Working Environment for the development of the smart contracts

A common working environment for Ethereum smart contract development typically involves a combination of tools, frameworks, and infrastructure that enable developers to create, test, and deploy smart contracts on the Ethereum blockchain. Here's an overview of the components we have used in our environment to develop the smart contracts:

1. **Integrated Development Environment (IDE)**: We have used popular IDEs like *Visual Studio Code*[3] and *Remix*[4]. These IDEs provide a user-friendly interface for writing, debugging, and deploying smart contracts. They offer features such as syntax highlighting, code completion, and integrated testing frameworks. *Visual Studio Code* has a version for *Windows*, *Linux*, and *macOS*, while *Remix* is a web application.

2. **Solidity Language**: *Solidity* is the primary programming language for Ethereum smart contract development. It is a statically typed language specifically designed for writing smart contracts. We have written Solidity code to define the logic and behavior of the smart contracts of the proposed protocols.

3. **Ethereum Client**: Developers need an Ethereum client to interact with the Ethereum network. Popular choices include *Geth*[5] and *Parity*[6]. These clients allow you to connect to the Ethereum network, create local test networks, deploy contracts, and interact with existing contracts. But to facilitate the interaction with the Ethereum network, we have used one of the simplest solutions: an *Infura*[7] node. *Infura* is a popular web service that provides developers with a reliable and scalable infrastructure to connect to the Ethereum blockchain. It acts as a gateway or access point to the Ethereum network, allowing developers to interact with the blockchain without having to run their own Ethereum node.

[3] https://code.visualstudio.com/
[4] https://remix.ethereum.org
[5] https://geth.ethereum.org/
[6] https://www.parity.io/technologies/ethereum/
[7] https://www.infura.io/

4. **Testing Frameworks**: Robust testing is crucial for smart contract development. Tools like *Truffle*[8] and *Hardhat*[9] provide testing frameworks that help developers write and execute unit tests, integration tests, and even simulate complex scenarios to ensure the reliability and security of the smart contracts. We have chosen Hardhat as our main framework. Hardhat is an open-source development environment and task runner specifically designed for Ethereum smart contract development. It offers a wide range of features and tools that help developers build, test, and deploy smart contracts more efficiently.

5. **Ethereum Network**: Developers deploy and interact with smart contracts on the Ethereum network. We have used the *Harhat* local test network, and real Ethereum test networks like *Goerli* and *Rinkeby*, for development and testing purposes. In some cases, we also have tested our solutions in the *Mumbai* test network, a test network specifically designed for Ethereum developers and users of the *Polygon* network. It is a Layer 2 scaling solution that aims to improve the scalability and performance of Ethereum-based applications.

6. **Web3 Libraries**: Web3 libraries, such as *Web3.js*[10] or *ethers.js*[11], provide APIs to interact with the Ethereum network. These libraries allow developers to send transactions, query contract data, and listen for events emitted by smart contracts. In this case, we have chosen *ethers.js*.

7. **Version Control System**: We have used *Git*[12], a commonly used software for version control to manage code changes, collaborate with team members, and maintain a history of the project. We also have used **GitHub**[13], an Internet hosting service for software development and version control using Git, to publish the repositories with the code of the proposed solutions.

8. **Security Tools**: To enhance smart contract security, developers may use tools *Slither*[14] to analyze code for vulnerabilities, perform static analysis, and detect potential security risks. We have used this software in some of the protocols.

In Figure 6.1 we have depicted our development configuration. It's important to note that the Ethereum ecosystem is continually evolving, and new tools and frameworks may emerge over time. Developers should stay updated with the latest developments and adapt their working environment accordingly.

To integrate the majority of these tools, we have used *Node JS*[15], a JavaScript runtime that allows developers to execute JavaScript code outside of a web browser. It provides a robust and scalable environment for server-side applications, making it a popular choice for Ethereum smart contract development. Functionalities provided by *Node JS* are implemented by independent modules and packages. In this way, we use NPM

[8] https://trufflesuite.com/
[9] https://hardhat.org/
[10] https://web3js.readthedocs.io/
[11] https://docs.ethers.org/
[12] https://git-scm.com/
[13] https://github.com/
[14] https://github.com/crytic/slither
[15] https://nodejs.org/

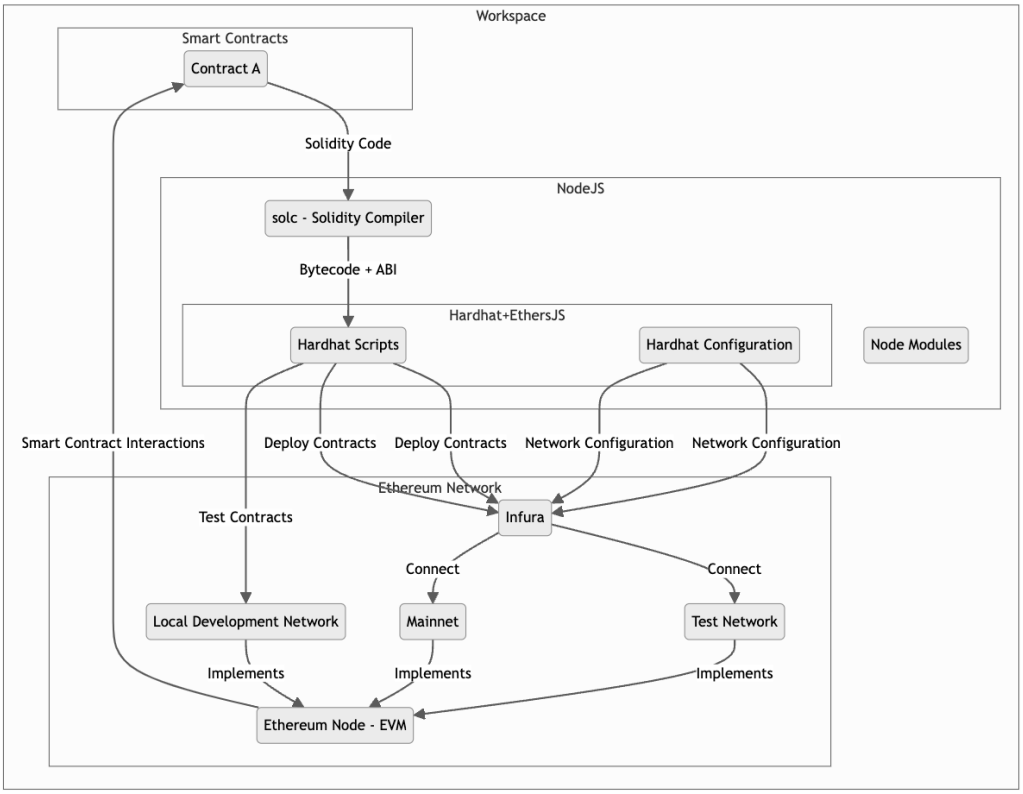


Figure 6.1: Smart contracts development architecture

(Node Package Manager)[16] to easily install/uninstall, configure and update the different modules and software packages of the platform (called third-party modules).

An important configuration file of a *Node JS* project is *package.json*. This is a JSON format file that is stored in the application root folder. This file provides the specific aspects to manage the module dependencies that the application requires. For instance, the file states information like our application name, package versions (name and version together work as an identifier that is assumed to be unique), license, directories, version control repository and so on. Keeping in mind this structure, a smart contract ready to deploy will be stored inside a folder within the solidity file that specifies the code of our application, the *package.json* and the node modules folder with all the necessary packages.

To see all the required modules in a repository, we can look at the *dependencies* section in the *package.json* file. Commonly packages used in a smart contract repository are:

- ***ethers.js***[17] is a powerful JavaScript library that provides a wide range of tools and utilities for interacting with the Ethereum blockchain. It offers a user-friendly and consistent API for working with smart contracts, sending transactions, and

[16] https://www.npmjs.com/
[17] https://www.npmjs.com/package/ethers

querying blockchain data. *ethers.js* supports multiple Ethereum networks, integrates well with *Node.js*, and simplifies tasks like contract deployment, function calls, and event listening.

- ***Hardhat***[18] is a development environment and task runner for Ethereum smart contract development. It offers a comprehensive suite of tools and features that streamline the development workflow. *Hardhat* supports tasks like contract compilation, testing, deployment, and script execution. It integrates with popular testing frameworks and provides an extensible plugin system, allowing developers to customize their development environment according to their specific requirements.

- ***hardhat-gas-reporter***[19] is a plugin for Hardhat that generates detailed gas consumption reports for smart contract transactions and function calls. It helps developers analyze and optimize gas usage in their contracts. The gas reporter plugin provides valuable insights into gas costs, allows for performance comparisons, and aids in identifying potential gas optimization opportunities.

- ***OpenZeppelin***[20] is a widely used library for developing secure and audited smart contracts on Ethereum. It provides a collection of reusable and battle-tested contracts that follow best practices for security and functionality. *OpenZeppelin* covers various areas like access control, token standards (*ERC-20*, *ERC-721*), payment channels, and more. By leveraging *OpenZeppelin*, developers can save time and reduce risks by utilizing pre-audited and community-reviewed smart contracts.

- ***Mocha***[21] is a widely adopted JavaScript testing framework that provides a flexible and feature-rich environment for writing test suites. It is commonly used in Ethereum smart contract development to write and execute unit tests and integration tests. *Mocha* offers powerful features such as test organization, test hooks, asynchronous testing support, and comprehensive reporting, making it a preferred choice for testing Ethereum smart contracts.

- ***Chai***[22] is a popular assertion library for JavaScript that is often used in Ethereum smart contract testing. It provides a readable and expressive syntax for defining assertions, making tests more concise and intuitive. *Chai* offers a variety of assertion styles (such as should, expect, and assert) and integrates seamlessly with testing frameworks like Mocha.

- ***circomlib***[23] is a library that provides cryptographic primitives and circuits for zero-knowledge proofs (ZKPs) on Ethereum. It includes functions and tools for building circuits using the *Circom* language, which is used to create arithmetic and boolean circuits for ZKPs. *circomlib* is commonly used for developing privacy-focused applications and protocols on Ethereum.

[18] https://www.npmjs.com/package/hardhat
[19] https://www.npmjs.com/package/hardhat-gas-reporter
[20] https://www.npmjs.com/package/@openzeppelin/contracts
[21] https://www.npmjs.com/package/mocha
[22] https://www.npmjs.com/package/chai
[23] https://www.npmjs.com/package/circomlib

- ***snarkjs***[24] is a JavaScript library for creating and verifying Zero-Knowledge Proofs (ZKPs) on the Ethereum blockchain. It provides a set of tools for working with Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zkSNARK) proofs. *snarkjs* allows developers to generate proofs, verify proofs, and perform other operations related to zero-knowledge proofs. It is commonly used in applications requiring privacy-preserving features, such as confidential transactions or identity management.

Inside the *package.json* file we also define the scripts of the project. Important scripts of a typical smart contract solution, using the *Hardhat* framework, are:

- **Compile**: This script compiles the Solidity smart contracts using the Solidity compiler. It may also include additional steps like cleaning the build artifacts before compilation.
- **Test**: This script runs the smart contract tests using a testing framework like *Mocha*. It allows an optional flag to track the gas usage of test cases, which can be useful for optimizing gas consumption.
- **Deploy**: This script deploys the smart contracts to a specified network, typically the local *Hardhat* development network by default. It compiles the contracts, deploys the desired contract using the deploy function from Hardhat's deployment module, and logs the deployed contract's address.

6.2 Architecture of the web interface

In some cases, we have also developed the web user interface to interact with the smart contracts deployed in a real test network like Rinkeby or Goerli. The architecture of a web3 application that interacts with Ethereum smart contracts typically involves multiple layers and components. Here's an overview of the architecture that we have used (see Figure 6.2):

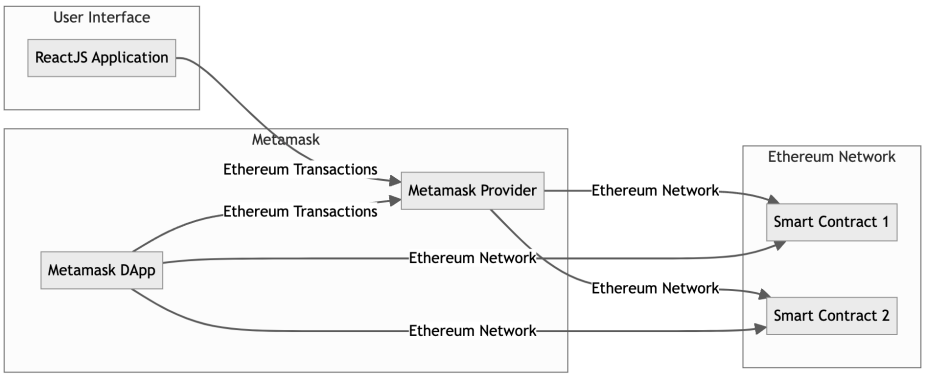


Figure 6.2: Web3 application user interface architecture

[24] https://www.npmjs.com/package/snarkjs

1. **User Interface (UI) Layer**

    - **React.js**[25]: React.js is a popular JavaScript library for building user interfaces. It provides a component-based approach for creating interactive and responsive UIs.
    - **React Router**[26]: React Router is a library for handling routing within a React application. It enables navigation between different views and components of the web3 application.

2. **Ethereum Integration Layer**:

    - **Metamask**[27]: Metamask is a browser extension wallet that allows users to interact with Ethereum-enabled applications. It provides a secure way to manage user accounts, sign transactions, and connect to the Ethereum network.
    - **ethers.js**: ethers.js is a powerful JavaScript library that serves as the bridge between the web application and the Ethereum network. It provides a set of APIs for interacting with smart contracts, sending transactions, and querying blockchain data. It also handles the communication with the Ethereum network through Metamask.

3. **Smart Contract Abstraction Layer**:

    - **Smart Contract ABIs**: Application Binary Interfaces (ABIs) are JSON representations of smart contracts that define the contract's functions, events, and data structures. ABIs are used to interact with smart contracts from the web application.
    - **Contract Instances**: Contract instances are generated from the smart contract ABIs using ethers.js. They represent the deployed instances of the smart contracts on the Ethereum network and provide methods for interacting with the contracts' functions and events.

4. **Ethereum Network**:

    - The **Ethereum network**, including testnets (like Rinkeby or Goerli) or the Ethereum mainnet, serves as the decentralized infrastructure where the smart contracts are deployed and transactions are executed.

The flow of the application typically involves the following steps:

- The React.js components render the UI, presenting information and allowing user interactions.
- When the user performs an action that requires interaction with a smart contract (e.g., calling a function, sending a transaction), the request is sent from the UI layer to the Ethereum Integration layer.

[25] https://www.npmjs.com/package/react
[26] https://www.npmjs.com/package/react-router
[27] https://metamask.io/

- Metamask, as a browser extension, provides a pop-up window for the user to review and confirm the transaction details, sign the transaction, and provide access to the user's Ethereum account.
- ethers.js, integrated with React.js, handles the communication with Metamask and the Ethereum network, sending the transaction or querying data from smart contracts.
- The Ethereum network processes the transaction or provides the requested data, and ethers.js receives the response.
- The UI layer updates with the results, displaying the appropriate information to the user.

By leveraging this architecture, developers can build web3 applications that seamlessly interact with Ethereum smart contracts using the Metamask wallet, the ethers.js library, and React.js components.

CHAPTER 7

REJECTABLE NFTS PROTOCOL

As detailed in subsection 4.6.1, the EIP-721 standard proposal defines the interface for Non-Fungible Tokens (NFTs). This type of token was designed to create unique and non-fungible tokens, with the impossibility of being damaged or destroyed. The EIP-721 standard has almost limitless applications: it can represent ownership of digital or physical assets, unlock access to certain services, and more. In short, it can represent any single object or right transferable between a sender and a receiver.

After analyzing the rejectability property of NFTs, we propose an improvement of the EIP-721 standard to enable selective transfers of tokens. Currently, NFTs can be transferred with the owner's consent (by themselves or by an authorized party), but the receiver cannot decline the reception of the token, which will be transferred to their wallet. Therefore, we propose an enhancement to the NFT standard that allows the rejection of token reception, creating a token usable in a secure way for applications requiring selective reception.

This improved protocol for the EIP-721 standard, allowing rejection of token reception, is instrumental in another protocol discussed within this thesis. Notably, it will be utilized in the "Two-steps Certified Notifications Protocol" described in Chapter 12. The development of Rejectable NFTs becomes essential for advancing this protocol, demonstrating its versatility and potential to enhance security and user control in various blockchain-based applications. This highlights its significance in scenarios where selective acceptance of digital assets is crucial.

7.1 Contribution

Regarding Blockchain technology, one of the most widespread uses of Ethereum or compatible blockchain networks is the use of the ERC-20 and ERC-721 token standards. The first represents a fungible token, interchangeable with each other because they all represent the same value, and the second represents a non-fungible token, unique and not interchangeable. Since these unique tokens can be transferred and the transfers are

recorded on the blockchain, they could be considered a valid medium of information transfer.

If we analyze the properties of Non-Fungible Tokens (NFTs), we can see that this standard lacks inherent rejectability, which limits control over unwanted or unsolicited transfers. Recognizing this gap, we propose the development of a new protocol, *Rejectable Non-Fungible Tokens (RejNFTs)*. This protocol, introduces the capability for recipients to accept or reject NFTs, thus enhancing user autonomy and security within the blockchain ecosystem.

A foundational concept in the field of fair exchange, particularly within the context of certified email exchanges, is addressed in one of the seminal papers by Kremer et al. [92]. This paper introduces the concept of *no author-based selective receipt*, emphasizing the necessity for fair exchanges in certified delivery systems. Fairness in this context is achieved when the recipient can make an informed decision about accepting or declining the notification or delivery, thereby providing non-repudiation of reception evidence.

The concept described by Kremer et al. is known as the **selective receipt property**, which allows recipients to decide whether to accept or reject the delivery based on certain criteria [92]. The paper distinguishes between two types of selective receipt: one based on the content of the message and another based on the sender's identity. It argues that selective receipt based on content is not feasible for a notification or delivery service, as it would undermine the integrity of the service. Conversely, rejection based on the sender's identity is possible, as recipients might infer the content of the message from the sender's identity, thus necessitating the concealment of the sender's identity until the recipient accepts the message.

This idea of selective reception of data has been used in several protocols since its definition. There are previous studies related to the selective reception of certified E-mail, like in [164] or [94]. These papers propose different solutions to the selective reception of E-mails, to guarantee the non-repudiation of origin and reception proofs.

The explicit and clear acceptance of the exchanged item in fair exchange protocols can lead a stronger protocol specifications with new interesting security features. We have applied one of the consequences of the idea introduced by Kremer et al. in a fair exchange of digital tokens by creating new evidence to reveal the receiver's commitment to accept the token. The introduction of this new evidence causes an interesting property in a token transfer that we have called: *discriminating or selective reception of tokens*. In other words, a receiver of a token has to be able to decide whether he wants to be the new owner of it or not. Since the existing standards for tokens do not offer this feature this thesis presents a new standard proposal.

Reviewing the Ethereum Improvement Proposals (EIPs) [1], that describe standards for the Ethereum platform, we can see that there does not exist an EIP that lets the receiver the possibility to cancel or reject the transfer of a token. There exists a proposal, the EIP-1404, "Simple Restricted Token Standard" [2], that is an easily extendable standard for issuing tokens with transfer restrictions. But if we see this proposal in detail, we can check that this EIP doesn't have the same goal as the one presented in the current thesis. The EIP-1404 proposal tries to help token issuers manage their

[1] https://eips.ethereum.org/

[2] erc1404.org/

compliance requirements, enforcing complex transfer restrictions. Some examples can be: restricting the maximum ownership of a single individual or entity, preventing token holders in a single jurisdiction from trading with token holders in another jurisdiction, letting the issuer revoke a token if necessary, etc...

Regarding the withdrawn EIP proposals we can find the following related EIP document:

- **EIP-875**. It proposes to reduce the cost of transfer transactions using a new method called **trade** that allows its processing in batches and thus to make a change of ownership by P2P.

Additionally, concerning the stagnant EIP proposals that have had more than six months of inactivity as of July 2022, we have identified the following related EIPs:

- **EIP-902** Token validation proposal. In this case, although the effect is not mentioned as applying to NFT tokens, they specifically mention a series of methods intended to validate the change of ownership. This concept should apply to Non-Fungible Tokens and we must define some mechanism to invalidate the change of ownership transaction.

- **EIP-926** Address metadata registry. In this case, an attempt is made to define a metadata registry, in which a specific attempt is made to define a mechanism to control "authorizations, acceptances of token content, and requests or complaints". An interesting aspect of this proposal is the implementation of an informal registry or issuing entity that allows such control, under a term called provider.

- **EIP-927**. Generalized authorizations. In this proposal there is no mention of a space or structure for metadata as in the previous case but, nevertheless, a structure similar to that of the supplier is defined. In this case, an external registry is proposed to validate the operations. This external agent is directly involved in the actions that affect the user. This concept applies directly to the ERC-20 standard instead of the NFT or ERC-721.

- **EIP-1056**. Low-weight identities. This concept involves an additional level of identity/property that can be interpreted similarly to the previous one, EIP-927. This proposal applies to ERC-20.

- **EIP-1080**. Recoverable tokens. This proposal discusses the options and mechanisms for undoing a change of ownership, allowing the generation of additional status. This proposal applies to ERC-20.

- **EIP-1132**. Ability to temporarily apply a lock to a token. In this case, a mechanism is defined to prevent actions with temporary character and that could be applicable to transfer request actions. However, its application is only for ERC-20.

- **EIP-2615**. NFT token with mortgage and rental functions. It introduces a new type of functionality that covers the approval of the operation and that is of application to the transfer of ownership.

In addition to that, concerning the draft EIP proposals, there are the following related EIPs documents:

- **EIP-4494**. Permit for ERC-721 NFTs. This proposal applies to NFTs and makes specific mention of a change of ownership with an additional permit that is without cost. It is based on ERC-2612 which applies to fungible tokens or ERC-20. In this sense, an additional method is generated that allows generating the permit and that is without associated cost. As mentioned, the `permit()` function would be sufficient to enable a `safeTransferFrom`, which in this case would allow accepting the change of ownership operation without additional cost.

- **EIP-5050**. Interactive NFTs with Modular Environments. This proposal, of a general and broad nature, explicitly specifies as a benefit the generation of additional action chains for the transmission of property, and, in this sense, a series of actions are allowed that must validate the transfer through the function `getApprovedForAction(address _account, bytes4 _action)` method and generating a state machine that could be applicable.

Finally, in the accepted EIP proposals (*final* status) there is the following EIPs document:

- **EIP-2309**. ERC-721 Consecutive Transfer Extension. Applicable to NFTs, this approved proposal allows a block of transfers or changes of ownership that allows to minimize the number of transactions in a block during minting. This process would allow the transfer of A - B, the denial of B (B - A) and, in such a case, an A-A result that would not generate any change of ownership.

In relation to the EIP proposals under review and last call, no proposals related to the denial of ownership have been found.

Thus, we can conclude that there are no proposals aimed at the change of ownership at no additional cost and considering a competent or legal authority. However, there are some proposals related to this concept for fungible tokens that could be adapted. In the case of NFTs, although there is no issue of the cost for such an avoided change of ownership (reject to NFT), we must highlight the proposal in draft quality EIP-4494 that would allow a transfer with lower cost. However, the proposal that we make in this chapter is innovative and, overall, does not exist as rejected to any effect as it is the case for similar proposals related to fungible tokens.

As a result of this analysis, as far as we know, it does not exist any previous work related to the use of Non-Fungible Tokens to add the possibility to reject a transfer of ERC-721 tokens thus there is no solution to use NFTs for the applications that require this property, as some of the protocols that will be presented in this thesis.

With this, after analyzing the properties of the NFTs, we have determined that these tokens would lack some useful security properties. This protocol is focused on one of these problems, the impossibility of rejecting the reception of a token. This problem is due to the fact that we can transfer both ERC-20 and ERC-721 tokens, but we can't perform a selective reception, because we can't reject these transfers. A rejectable token would allow the selective reception of tokens.

To visualize the importance of the selective rejection of tokens we will see two examples of applications that can be benefited from this property.

The first is the case of certified notifications or eDeliveries. In section 5.1, we presented some use cases for certified notifications. These kinds of notifications provide users with both evidence that the sender has sent the intended message and evidence that the data was delivered or, at least, the delivery attempt was made. If a selective reception of ERC-721 tokens could be made, that is, allowing the receiver to reject their transfer, the problem of sending tokens as notifications could be solved. Another use case is the use of tokens to represent a passive value and not an active one, such as Non-Fungible Tokens (NFTs) that represent a "negative value" asset, like a loan, a burden, the membership to groups where we do not want to be, any kind of liability, etc... In this case, it is evident that a receiver must approve the reception of the token.

Since there is no Ethereum Improvement Proposal (EIP) that allows us to reject the transfer of an ERC-721 token, this protocol proposes a new standard that allows selective reception of this type of token, implementing the possibility of rejecting them.

7.2 Protocol design

To let receivers of an ERC-721 token reject the transfer, we must add certain functions and events to the code of a regular NFT. In our proposal, we have a new mapping, `_transferableOwners`, that will store the owner to whom we want to transfer the token. With this, when we transfer the NFT, instead of directly transferring the ownership, we add the address of the receiver, for that NFT id, to the `_transferableOwners` mapping.

Then, the receiver will be able to accept or reject the transfer. Besides, we have introduced another alternative: if the receiver hasn't yet accepted or rejected the transfer, the sender will still have the chance to cancel the transfer by removing the receiver from the `_transferableOwners` mapping. This new mapping stores the proposed owners of the tokens and it is defined in Listing 3.

We also need to take into consideration the `mint()` function, where the NFT is created and doesn't have yet an owner. In fact, minting an NFT represents also a transfer of ownership, from the zero address to the receiver.

In Figure 7.1 there is a state diagram of the proposed protocol, where we can see the states of the exchange of the NFT between the sender and the receiver. Let **A** be the creator of an NFT and let **B** be the intended receiver of it. Then:

- **A** creates the token executing the operation `A.mint(B)`. With this operation, the address of B is introduced in `_transferableOwners`.
- Now **B** can accept the transfer by executing `B.acceptTransfer()` or, alternativelly, it can reject the NFT with `B.rejectTransfer()`.

```
// Mapping from token ID to transferable owner
mapping(uint256 => address) private _transferableOwners;
```

Listing 3: transferableOwners mapping

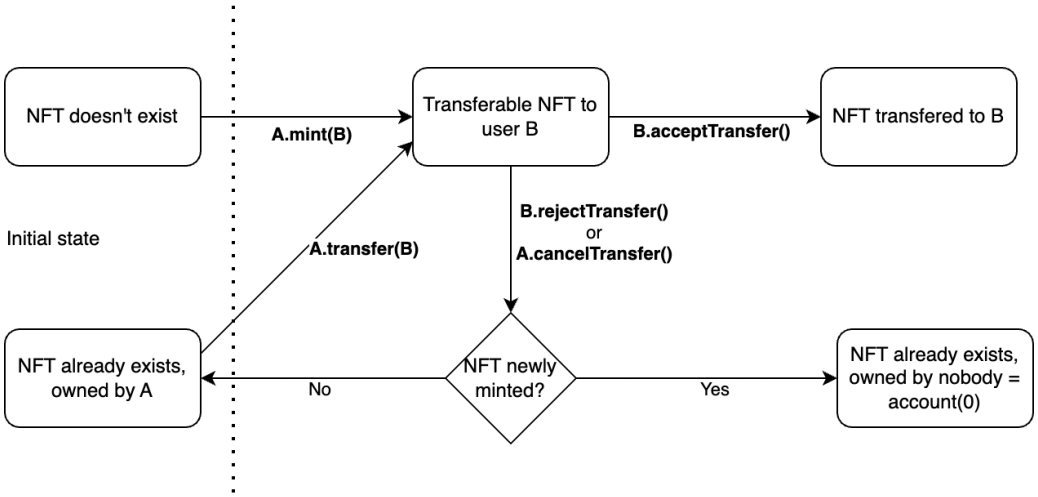


Figure 7.1: States of the RejectableNFT protocol

- The procedure is similar if the NFT was already minted (**A** simply changes the minting operation by a transfer `A.transfer(B)`).
- In addition to that, as we have mentioned above, if **B** has not yet accepted the transfer, **A** has the ability to cancel the transfer by executing `A.cancelTransfer()`.

To enable the rejection of an ERC-721 token, apart from the introduction of the `_transferableOwners` mapping, we also need to modify the private functions `_mint()` and `_transfer()`, that will add the proposed receiver of the token to the mapping `_transferableOwners`, instead of directly transfer the NFT. These private functions are called from the following functions of the ERC-721 standard:

- `safeTransferFrom()`
- `transferFrom()`
- `mint()`

These public functions already check that the only accounts that can transfer a token are its owner or the corresponding `approved` or `approvedForAll` accounts. To implement this additional feature of the standard, we also need these new three events, that will be emitted from the corresponding functions:

- `TransferRequest`: emitted when a token is proposed to be transferred.
- `RejectTransferRequest`: emitted when the receiver rejects the transfer of the token.
- `CancelTransferRequest`: emitted when the sender cancels the transfer of the token.

The `Transfer` event, that already exists in the current ERC-721 implementation, will be emitted when the transfer is really performed, changing the ownership from the sender to the receiver.

With this, the code of the private functions `_mint()` and `_transfer()` is described in Listing 4

```
function _mint(address to, uint256 tokenId) internal
        virtual {
    require(to != address(0), "ERC721: mint to the zero
        address");
    require(!_exists(tokenId), "ERC721: token already
        minted");

    _beforeTokenTransfer(address(0), to, tokenId);

    _transferableOwners[tokenId] = to;

    emit TransferRequest(address(0), to, tokenId);

    _afterTokenTransfer(address(0), to, tokenId);
}

function _transfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual {
    require(RejectableNFT.ownerOf(tokenId) == from,
        "ERC721: transfer from incorrect owner");
    require(to != address(0), "ERC721: transfer to the
        zero address");

    _beforeTokenTransfer(from, to, tokenId);

    // Clear approvals from the previous owner
    _approve(address(0), tokenId);

    _transferableOwners[tokenId] = to;

    emit TransferRequest(from, to, tokenId);

    _afterTokenTransfer(from, to, tokenId);
}
```

Listing 4: Modified mint and transfer Functions

There are hooks like the `beforetokenTransfer()` hook that allow lots of flexibility in modifying the behavior of a token before the token is transferred. This will let us restrict transfer to only registered candidates, or to only allow tokens that are already accepted by the receiver. But it doesn't help us to do what we want to do, because in our proposed protocol the first step must be the call to the transfer function by the sender, and to use this hook, we would need a first step of the receiver calling an acceptance function.

In addition to the three events described above, in order to complete the protocol we also need to implement the three new functions:

- `acceptTransfer()`: the receiver can call this function to accept the transfer proposal made by the sender. We need to check that the receiver is included in the `_transferableOwners` mapping.
- `rejectTransfer()`: the receiver can call this function to reject the transfer proposal made by the sender. Once again, this can be only done by the proposed receiver of the token.
- `cancelTransfer()`: the sender of the transfer proposal can cancel it by calling this function, if the receiver hasn't still called the `acceptTransfer()` or `rejectTransfer()` functions. This function can also be called by the minter of the token, if the token is new and it still hasn't an owner.

The code of these functions is specified in Listing 5

7.3 Implementation

All the code of the Rejectable Non-Fungible Token (RejNFT) can be found in the *rejectable-nft*[3] repository of the Security and e-Commerce (SECOM) Research Group from the University of the Balearic Islands. This repository has the code of a simple ERC-721 token, the code of the new **RejectableNFT** token, and the necessary scripts to compile, deploy and test the cited smart contracts, comparing the gas cost of the current NFT code and the proposed one.

7.4 Security properties analysis

NFT adoption requires the definition of exchange protocols on a Peer-to-Peer (P2P) network. The first NFT projects were born thinking on an owner-buyer basis. However, with the Ethereum blockchain and the release of the ERC-721 standard, NFTs expanded to many applications. For example, NFTs are used for digital art, fashion, certifications, gaming, virtual events, collectibles, healthcare and so on [76, 165]. With ERC-721 standard, every single token is unique and identified by a `uint256` variable called `tokenId` in its context. But, to globally identify a NFT we have to use the pair of *contract address* and `tokenId`. Later, the standard ERC-1155 [166] was introduced, which extends the representation of ERC-20 and ERC-721 tokens. This is a multi-token standard where tokens are grouped into different types and the tokens of the same type are fungible.

Despite the different standards and many NFT applications, the logic between owner and buyer in a token transfer has not been overcome. Hence, the standards specify an active owner who transfers the token to a passive buyer, thus any transfer cannot be rejected by the receiver. However, not only NFTs can represent assets (goods and rights of a company or other economic entity) but also they can represent liabilities (obligations and debts) that not always the receiver wants to assume. Also, for

[3] https://github.com/secomuib/rejectable-nft

```
function acceptTransfer(uint256 tokenId) public {
    require(_transferableOwners[tokenId] == _msgSender(),
        "RejectableNFT: accept transfer caller is not the
        receiver of the token");

    address from = RejectableNFT.ownerOf(tokenId);
    address to = _msgSender();

    if (from != address(0)) { // Perhaps previous owner is
            address(0), when minting
        _balances[from] -= 1;
    }
    _balances[to] += 1;
    _owners[tokenId] = to;

    // remove the transferable owner from the mapping
    _transferableOwners[tokenId] = address(0);

    emit Transfer(from, to, tokenId);
}

function rejectTransfer(uint256 tokenId) public {
    require(_transferableOwners[tokenId] == _msgSender(),
        "RejectableNFT: reject transfer caller is not the
        receiver of the token");

    address from = RejectableNFT.ownerOf(tokenId);
    address to = _msgSender();

    _transferableOwners[tokenId] = address(0);

    emit RejectTransferRequest(from, to, tokenId);
}

function cancelTransfer(uint256 tokenId) public {
    //solhint-disable-next-line max-line-length
    require(
      // perhaps previous owner is address(0), when minting
      (RejectableNFT.ownerOf(tokenId) == address(0) &&
          owner() == _msgSender()) ||
      _isApprovedOrOwner(_msgSender(), tokenId),
      "ERC721: transfer caller is not owner nor approved");

    address from = RejectableNFT.ownerOf(tokenId);
    address to = _transferableOwners[tokenId];

    require(to != address(0), "RejectableNFT: token is not
        transferable");
    _transferableOwners[tokenId] = address(0);

    emit CancelTransferRequest(from, to, tokenId);
}
```

Listing 5: New functions of the Rejectable NFT protocol

instance, a token can represent a notification that the receiver may not want to receive for temporary reasons (e.g. he/she is now on vacation) or for permanent reasons (e.g. the receiver is no longer interested in this type of topic).

Therefore, the introduction of our *Rejectable NFT*s can provide new security properties to the ones introduced by ERC-721 tokens. Actions concerning data managed in a blockchain provide specific data features, such as transparency, immutability and consistency [167]. *Rejectable NFTs* are a blockchain-based technique and, for this reason, offer the following security properties thanks to the specific blockchain features mentioned above and the protocol described in this chapter:

- **Accountability and Verifiability**: The NFT, the metadata associated with the token and the ownership of the token can be publicly verified because all the operations concerning the tokens are conducted through smart contracts. Therefore, minting and transferring NFTs actions are publicly accessible. This property ensures **Authenticity** and **Non-Repudiation of Origin** since the transfer action is triggered by a signed confirmed transaction on the blockchain that invokes the smart contract that manages the NFTs.

- **Integrity and Availability**: the NFT metadata and ownership are added to a new block on the blockchain after a minting or trading action. As a result, the history of the NFT and its associated data remains tamper-resistant, and they are always available since blockchain is considered a persistent storage that never goes down (i.e. if at least one node is in the blockchain network).

- **Transferability and No-overspending**: Every single NFT can be arbitrarily transferred, traded or exchanged by its owner. The data immutability and consistency ensured by the distributed consensus make visible to all future data manipulations and, establish a single truth across the blockchain network. In addition to that, the consensus mechanism prevents any overspending cases because miners can check the entire history of every transaction before validating it.

- **Non-repudiation of reception and Selective Receipt:** With the introductions of our proposal in section 7.2 any transfer action switches the token to a standby state until the receiver accepts or rejects it (see Figure 7.1). Thus, the receiver can decide whether to accept the token or not on a particular basis. That creates what is called a transfer with *Selective Receipt*. Moreover, the acceptance transaction generates a *non-repudiation of reception evidence*.

7.5 Performance analysis

In Figure 7.2 we can see the output of the gas cost analysis of the functions described in section 7.2. This report has been generated by the *hardhat-gas-reporter*[4] plugin, executed into the *hardhat*[5] Ethereum development environment. This gas cost is calculated using the Ether (ETH) price of day 24/07/2022, 1604.46 USD/ETH.

[4] www.npmjs.com/package/hardhat-gas-reporter
[5] hardhat.org/

Solc version: 0.8.6		Optimizer enabled: true		Runs: 200	Block limit: 30000000 gas	
Methods		6 gwei/gas			1604.46 usd/eth	
Contract	**Method**	Min	Max	Avg	**# calls**	**usd (avg)**
NFT	safeMint	-	-	95689	3	0.92
NFT	transferFrom	-	-	57575	1	0.55
RejectableNFT	acceptTransfer	51929	68647	66259	7	0.64
RejectableNFT	cancelTransfer	26955	28522	27739	2	0.27
RejectableNFT	rejectTransfer	-	-	26189	2	0.25
RejectableNFT	safeMint	-	-	75426	9	0.73
RejectableNFT	transferFrom	-	-	53937	4	0.52
Deployments					**% of limit**	
NFT		-	-	1353505	4.5 %	13.03
RejectableNFT		-	-	1539173	5.1 %	14.82

Figure 7.2: Gas cost of the RejectableNFT smart contract

Table 7.1: Comparison of the gas cost of the deployment of the TraditionalNFT and RejectableNFT smart contracts

TraditionalNFT	RejectableNFT	Difference
1353505	1539173	+13.72%

Table 7.2: Comparison of the gas cost of the functions of the TraditionalNFT and RejectableNFT smart contracts

Function	TraditionalNFT	RejectableNFT	Difference
safeMint()	95689	75426	-21.18%
transferFrom()	57575	53937	-6.32%
acceptTransfer()		66259	
rejectTransfer()		26189	
cancelTransfer()		27739	

If we compare the cost of the deployment of the smart contracts in Table 7.1, we can see that deploying the new **RejectableNFT** smart contract is 13.72% greater than deploying an ERC-721 token. This is due to the added code in section 7.2.

And regarding the gas cost of the functions, in Table 7.2 we can see that the minting is 21.18% cheaper in the **RejectableNFT**, and the transfer is 6.32% cheaper. But we must take into consideration that the transfer in the **RejectableNFT** smart contract is only a transfer proposal, and the receiver will need to accept the transfer by calling the `acceptTransfer()` function. This supposes that the real transfer of ownership of the token is more expensive in the new standard, but the costs are distributed between the sender and the receiver.

The functions `acceptTransfer()`, `rejectTransfer()`, and `cancelTransfer()` have gas costs that cannot be compared to those of a traditional ERC-721 token, because these functions don't exist in the previous standard.

As a summary, we can say that the gas cost of both standards is not worth mentioning, because they are within normal terms of cost increase due to the added code, so we can state that the new proposal is worth it due to the benefits obtained by this little increase in terms of cost.

7.6 Conclusions

In this chapter, we have presented a proposal to improve the ERC-721 standard to include a selective reception of the NFT by the receiver. With this, we achieve a non-repudiation of origin and reception proof when we send an NFT. The non-repudiation of reception proof can be especially useful when we send an ERC-721 token that doesn't represent an active, but a passive, a burden or any liability. The new proposal maintains the security properties of the ERC-721 standard and adds new properties to them, as it has been depicted in section 7.4.

The proposed protocol doesn't imply a big increment in gas cost terms, as we have demonstrated in section 7.5. The gas cost is the best measure of performance that we can have in blockchain networks like Ethereum, and the gas cost analysis demonstrates that adding some new functions and variables to achieve the capability to reject a NFT transfer doesn't necessarily imply a high increase in this measure.

As future work, we can continue with different improvements to the proposed standard:

- Let the sender propose to send the token to a member of a set of more than one receiver. In the end, only one of the members of the set will receive the NFT.
- Set a timeout for the transfer proposal. If the receiver doesn't accept the transfer in a certain amount of time, the transfer is automatically canceled.

This new protocol also leads us to other improvements, such as applying the same rejectable transfer protocol to ERC-20 or ERC-1155 tokens or implementing the selective receipt implementing other kinds of algorithms, such as blocking the transfer and using the `permit()` function described in EIP-4494[6].

Once this protocol is fully developed, it can be leveraged in subsequent protocols described in this thesis. Specifically, by utilizing the selective reception capabilities of the new protocol, we can simplify the certified notifications process in the protocols discussed in the next chapters, reducing operational complexity as detailed in Chapter 12. This enhancement not only demonstrates the practical applicability of the rejectable transfer feature but also underscores its potential to improve the efficiency and user experience of blockchain-based transactions.

[6] eips.ethereum.org/EIPS/eip-4494

CHAPTER 8

TWO-PARTY CERTIFIED NOTIFICATIONS PROTOCOL

In this chapter we introduce a protocol in the realm of Blockchain technologies, focusing on the provision of fair certified notifications, a topic previously analyzed in section 5.1. Certified notifications are one of the applications that require a fair exchange of values: a message and non-repudiation of origin proof in exchange for a non-repudiation of reception evidence. We propose two solutions that allow sending certified notifications when confidentiality is required or when it is necessary to register the content of the notification, respectively. First, we present a protocol for Non-Confidential Fair Certified Notifications that satisfies the properties of strong fairness and transferability of the proofs thanks to the use of a smart contract and without the need for a Trusted Third Party. Then, we also present a DApp for Confidential Certified Notifications with a smart contract that allows a timeliness optimistic exchange of values with a stateless Trusted Third Party.

8.1 Contribution

As discussed in Chapter 4, blockchain technology provides an immutable data registry that supports innovative solutions for traditional services. An example of such a service benefiting from blockchain's unique features is certified notifications, which enable senders to prove that they have sent a message to a receiver or set of receivers. This service ensures receivers have been granted access to messages from a definite point in time.

Certified notifications, along with other electronic services, such as the electronic signature of contracts, electronic purchase (payment in exchange for a receipt or digital product) or certified mail, require a fair exchange of items between two or more users. In order to create protocols that allow carrying out these exchanges and, at the same time, maintain the security of communications, there are solutions that fall into the generic pattern named *fair exchange of values*. A fair exchange always provides equal

treatment of all users, and, at the end of each execution, either each party has the element she wishes to obtain from the other party, or the exchange has not been carried out successfully for anyone (any party has received the expected item). In a typical notification case, the element to be exchanged is the message along with non-repudiation proof of origin and reception.

Most existing fair exchange protocols [81, 82, 83] rely on TTPs to mediate and resolve any disputes that emerge from transaction interruptions or fraudulent activities. These protocols also employ non-repudiation methods to create evidence documenting the actions of the participants involved. With the introduction of blockchain technology and smart contracts, there is potential to either substitute or enhance the role of TTPs with these innovations. This shift opens up new opportunities for devising more effective digital fair exchange solutions that adhere to the principles of equitable value exchange. An innovative approach within the Bitcoin network for creating fair exchanges involves encouraging participants to fulfill their end of the agreement by implementing financial incentives or penalties for those who act dishonestly [84, 85, 86].

In addition to that, the Ethereum blockchain and its cryptocurrency Ether (ETH) offers an even richer functionality set than conventional cryptocurrencies such as bitcoin, since they support smart contracts in a fully distributed system that could lead, as we will see in this chapter, to enable fair exchanges of tokens without the involvement of a TTP (since smart contracts are self-applied and reduce the need for trusted intermediaries or reputation systems that decrease transaction risks). This new technology allows us to define transactions with predetermined rules (written in a contract) in a programmable logic that can guarantee a fair exchange between parties with an initial mutual distrust. This feature prevents parties from cheating each other by aborting the exchange protocol and discharges the need for intermediaries with the consequent reduction of delays and commissions for their services.

The revealing power of the blockchain is further enhanced by the fact that blockchains naturally incorporate a discrete notion of time, a clock that increases each time a new block is added. The existence of a trusted clock/register is crucial to achieve the property of fairness in the protocols. This feature can make the cryptography model in the blockchain even more powerful than the traditional model without a blockchain where fairness is very difficult to guarantee without the intervention of a TTP.

Previous studies on fairness using blockchain have focused on fair purchase operations between a product (or a receipt) in exchange for cryptocurrencies (usually bitcoin) [99, 105, 100, 101]. [168], for example, uses a smart contract for the resolution of a purchase operation while [104] uses smart contracts and trusted execution environments to guarantee the fair exchange of a payment and the result of an execution.

In this chapter, we present two Blockchain-based Systems for Fair Certified Notifications. The first proposal facilitates a non-confidential fair exchange of a notification message for a non-repudiation of reception token without the involvement of any TTP, while the second enables a confidential fair exchange of a notification for a non-repudiation of reception token with the optimistic intervention of a stateless TTP. These protocols demonstrate how blockchain technology and smart contracts can introduce a new paradigm to address the fair exchange problem, potentially reducing or even eliminating the need for TTPs within such protocols.

8.2 Protocol design

In this section, we will analyze the possibilities of the use of blockchain-based technologies to provide a DApp for fair certified notifications reducing the involvement of trusted third parties compared with traditional approaches. We present a high-level description of two solutions (the details of them will be presented in subsection 8.3.1 and subsection 8.3.2, respectively). One of them is well suited for those notifications that do not require the confidentiality of the message (or even it is required that the message can be public and accessible to everybody). The other solution allows the message to be hidden from others than the receiver. As it is shown in the descriptions, in the first approach there is no need for a TTP in any step of the exchange nor in a dispute resolution phase while in the second proposal, the TTP will be involved only in the dispute resolution phase (optimistic protocol). Moreover, it is not required that this TTP stores information on the state of any transaction.

8.2.1 Non-confidential Certified Notifications without TTP

In this first proposal, we consider that confidentiality is not required or even desired. The sender executes the first step of the protocol using the DApp to register the hash of the notification message on the blockchain. At this point, the receiver does not have access to the message, although the transaction remains stored in the blockchain due to the fact that the registered value is the hash of the message and not the message itself.

The sender will make a new transaction including the message in a third step, provided that the receiver would have made a previous transaction manifesting his desire to receive the notification.

The protocol for non-confidential certified notifications has the states depicted in Figure 8.1, and works as follows:

1. The sender, who originates the message, uses the smart contract to publish in the blockchain the hash of the notification message. Other parameters of this

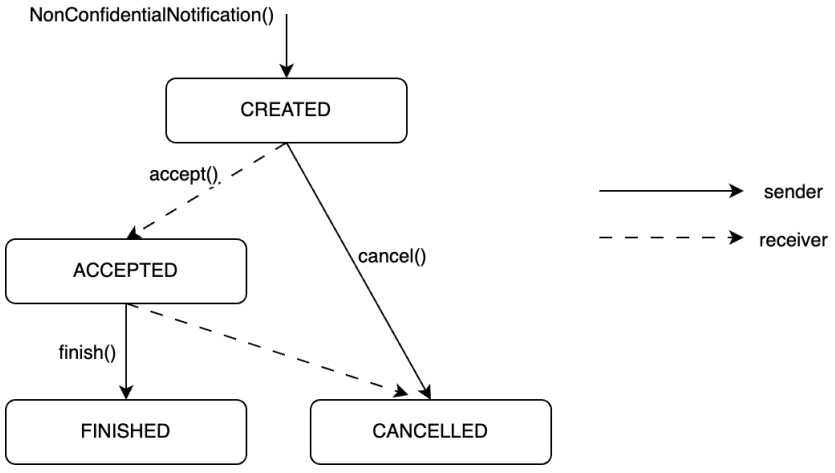


Figure 8.1: States of the non-confidential two-party certified notifications protocol

transaction are the address of the receiver and the deadline for the notification to be completed. Moreover, a deposit can be required in this step. The amount will be included in the transaction.

2. The receiver, if he accepts the reception of the notification, publishes a message expressing his will.

3. Finally, before the expiration of the deadline, the sender can execute the *finish* procedure to publish the message. As a consequence, the smart contract publishes the non-repudiation proof. If the execution of the exchange requires a deposit, the smart contract returns the amount to the sender.

   After the deadline, if the three steps have not been executed properly, the state of the exchange is not *finish* and then both parties can access a function in the smart contract to request the cancellation of the transferred elements.

   a) Cancellation of reception, requested by the receiver if the sender does not publish the message when the receiver has accepted the notification.

   b) Cancellation of delivery, requested by the sender, if the receiver has not accepted the notification.

   In both cases, the smart contract checks the identity of the user and the deadline. The smart contract generates a transaction to point out that the notification has been canceled. In the first case, the sender will not receive the refund of the deposit (this way, the deposit is useful to motivate the sender to finish the exchange before the deadline).

Since the message is included in a transaction, it will be registered in the blockchain, so the notification in this case is not confidential. This protocol is executed entirely over Ethereum, so no off-chain communication between the parties is required. This way, there is no need for communication channels between the parties.

8.2.2 Confidential Certified Notifications with Stateless TTP

This second proposal has been designed to take into account those notifications that require confidentiality. That is, the blockchain has to preserve the fairness of the exchange but the message cannot be stored in a publicly accessible block. The main difference with the first proposal is that in this case, the protocol allows an optimistic exchange, that is, the exchange can be executed completely without the intervention of the TTP nor the blockchain. Another important feature is that this proposal does not require a deadline and can be finished at any moment. A stateless TTP can be used to resolve the disputes that can arise between the parties.

The proposed protocol for confidential fair certified notifications is based on the protocol described in [91], an optimistic protocol in three steps with a trusted third party that is involved only in case of disputes between the parties. In [91] both parties can contact the TTP who maintains state information. The three steps of [91] are:

1. The sender A encrypts the message M with a symmetric key K, producing a ciphertext c. The key K is encrypted with the public key of the TTP (it means that

the TTP, who knows the correspondent private key, can decrypt it), producing K_t. A third element h_A is the signature of A on the concatenation of the hash of ciphertext c and K_t, part of the evidence of Non-Repudiation of Origin for B. Then A sends the triplet c, k_t and h_A.

2. B sends h_B, a signature of B on the concatenation of the hash of ciphertext c and k_t, evidence of Non-Repudiation of Receipt for B, to A.

3. A sends k_A, the key K enciphered with the private key of A, the second part of the Non-Repudiation of Origin evidence for B.

During this three-step protocol, the parties exchange the Non-Repudiation proofs together with elements that are useful in case of interruption of the exchange. These elements, as K_t, are managed by the TTP during a dispute resolution subprotocol. The execution of the dispute resolution subprotocol can be requested by both A and B contacting the TTP.

In this new blockchain-based solution, the originator A and the recipient B will exchange messages and non-repudiation pieces of evidence directly. Only as a last resort, in the case they cannot get the expected items from the other party, the smart contract or the TTP would be invoked, by initiating the *cancel* or *finish* functions. In comparison with the protocol described in [91], in the blockchain-based solution the role of the TTP has been reduced. The sender will never contact the TTP. The TTP will answer only requests from the receiver B by accessing the smart contract that has been deployed. The TTP is totally stateless and, in any case, it stores information about the state of any exchange.

The states of the protocol for confidential certified notifications are depicted in Figure 8.2, and the protocol works as follows: The parties, A (the sender) and B (the receiver) will execute a direct exchange in three steps, using the DApp (the details of it will be presented in subsection 8.3.2).

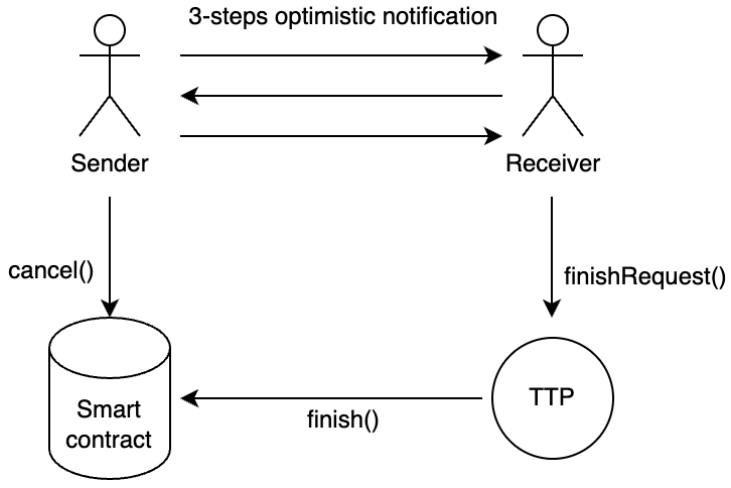


Figure 8.2: States of the confidential two-party certified notifications protocol

1. *A* sends an encrypted message to *B* using a session key. Moreover, *A* also sends an element to *B* that could be useful in case of dispute, that is, if *A* does not follow the steps of the protocol (i.e.: the session key encrypted with the public key of the TTP). The TTP is responsible for the deployment of the smart contract that will manage the exchange.

2. *B* sends the non-repudiation proof.

3. A sends the key to decipher the message.

   If some party does not follow the protocol, the exchange can be resolved as follows:

   a) If *A* does not receive the element of step 2., she can send a request to the smart contract. If the state of the notification is 'Created' (neither 'Canceled' nor 'Finished'), then the state will be changed to 'Canceled', indicating that the notification has not been performed successfully.

   b) If *B* does not receive the message in step 3, *B* will contact the TTP providing the received elements in step 1 together with the non-repudiation of reception proof. The TTP will access the smart contract to check the state of the notification. If the notification has not been canceled, the TTP will publish in the blockchain a non-repudiation of reception proof and the required elements for *B* to obtain the confidential message.

8.3 Implementation

The code of the non-confidencial[1] and confidential[2] smart contracts described in this section can be accessed via the GitHub repositories maintained by the Security and e-Commerce (SECOM) Research Group from the University of the Balearic Islands. You can review the development environment details in Chapter 6.

8.3.1 Non-confidential Certified Notifications without TTP

We have designed and implemented the smart contract for non-confidential notifications. For this proposal, a new instance of the smart contract will be created by the sender and it will manage all the steps of the exchange. It has been programmed using Solidity and it has been deployed over an Ethereum network. Ethereum addresses have been assigned to both the sender *A* and the receiver *B*. Both *A* and *B* will interact with the blockchain using Web3.js interfaces.

Listing 6 shows the smart contract that manages the non-confidential notifications. The constructor of the smart contract includes variables to store the addresses of the sender and the receiver, the hash of the notification message, the instant of execution of the first step of the exchange and the value of the execution period (the deadline). The contract includes five functions: to initiate the notification (the constructor), to accept a notification, to deliver the message, to cancel the exchange and to inquire about the state of the exchange. The certified notification is created by the sender, who specifies

[1] https://github.com/secomuib/NonConfidentialCertNotification/tree/master/contracts_2party
[2] https://github.com/secomuib/ConfidentialCertNotification/tree/master/contracts_2party

```
pragma solidity ^0.4.24;

contract Notification {
    address public sender; // Parties involved
    address public receiver;
    bytes32 public messageHash; // Message
    string public message;
    uint public term; // Time limit (in seconds)
    uint public start; // Start time
    enum State {created, canceled, accepted, finished }
    State public state;

    event StateInfo(State state);

    constructor(address _sender, address _receiver, bytes32
        _messageHash, uint _term) payable {
        require (msg.value>0); // sender deposit
        sender = _sender;
        receiver = _receiver;
        messageHash = _messageHash;
        start = now; // now = block.timestamp
        term = _term;
        state = State.created;
        emit StateInfo(state);
    }
    function accept() public {
        require (msg.sender==receiver && state==State.created);
        state = State.accepted;
        emit StateInfo(state);
    }
    function finish(string _message) public {
        require(now < start+term); // check deadline
        require (msg.sender==sender && state==State.accepted);
        require (messageHash==keccak256(_message));
        message = _message;

        // Sender receives the refund of the deposit
        sender.transfer(this.balance);
        state = State.finished;
        emit StateInfo(state);
    }
    function cancel() public {
        require(now >= start+term); // check deadline
        require((msg.sender==sender && state==State.created) ||
                (msg.sender==receiver && state==State.accepted));
        if (msg.sender==sender && state==State.created) {
            // Sender receives the refund of the deposit
            sender.transfer(this.balance);
        }
        state = State.canceled;
        emit StateInfo(state);
    }
}
```

Listing 6: Non-confidential two-party certified notifications smart contract

the receiver, the hash of the message and the maximum duration of the exchange. The smart contract also has an attribute to store the content of the message.

The contract also manages a Solidity event to follow the progress of the exchange. This event *stateInfo* allows the parties to see the evolution of the state of the exchange (*created*, *accepted*, *finished* or *canceled*).

Functions *Accept* and *Finish* check the identity of the address that throws the transaction. The address of the receiver and the sender are verified before updating the state. The function *Cancel* also checks the addresses. In this case, both the sender and receiver can execute the function, depending on the state of the exchange. Moreover, all the functions that can cause any change in the state of the exchange check the value of the variable *State*. The function *Finish* requires that the present time does not exceed the deadline before executing its code. Also, the function *Cancel* verifies that the current time is greater than the deadline before carrying on with the execution of it.

The smart contract will manage the publication on the blockchain of all the values of the required variables to maintain the fairness of the exchange following the protocol described in subsection 8.2.1.

8.3.2 Confidential Certified Notifications with Stateless TTP

We have designed and implemented a DApp that allows the optimistic exchange between the parties and a smart contract for the resolution of disputes. The smart contract has been programmed in Solidity and deployed over the Ethereum network. Ethereum addresses have been assigned to the sender A, the receiver B and the TTP. In comparison with the protocol described in [91], the role of the TTP has been reduced. The sender will never contact the TTP. The TTP will answer only requests from the receiver B by accessing the smart contract that has been deployed. The TTP is totally stateless and, in any case, it stores information about the state of any exchange. Both A and B can interact with the blockchain if it is necessary through the Web3.js interface. For this reason, we have also designed a web service where the web client can connect using TLS protocol. This web service is used on the off-chain communication exchanges between sender and recipient described in the protocol. In order to implement the cryptographic operations, we have used Stanford Javascript Crypto Library[3]. This enables us to use AES for the symmetric encryption operation, EC-ElGammal for the asymmetric encryption operations and ECDSA for the signature functions. However, the implementation of a PKI and the secure exchange of public keys are beyond the scope of this work. Listing 7 shows the smart contract that will manage the possible disputes between the parties after the execution of the exchange described in subsection 8.2.2 for confidential notifications. This smart contract is deployed by the TTP, which defines the identities of the sender and the receiver. The smart contract manages the variable *state* in order to keep track of the state of each exchange.

The event *stateInfo* allows the tracking of the evolution of each exchange state. The function *Cancel* checks the identity of the address that throws the transaction, which must be the address of the sender, together with the value of the variable *state*. This function can be executed only by the sender. The function *Finish* checks the identity of the party that sends the transaction, that is, the TTP, together with the value of the

[3] http://bitwiseshiftleft.github.io/sjcl/

```
pragma solidity ^0.4.11;

contract ConfidentialNotifications {
    //Parties involved
    address sender;
    address receiver;
    address ttp;

    string hB; //NMR proof
    string hBt; //Intervention proof
    enum State { created, canceled, finished }
    State public state;

    event stateInfo(State state);

    function ConfidentialNotification(address _sender, address
        _receiver){
        ttp = msg.sender;
        sender = _sender;
        receiver = _receiver;
        state = State.created;
    }
    function Cancel() returns (string) {
        if(msg.sender==sender){
            if(state==State.created){
                state=State.canceled;
                //return abort token
                stateInfo(state);
            }else if (state == State.finished){
                return hB;
            }
        }
    }
    function Finish(string _hB, string _hBt) returns (State) {
        if (msg.sender==ttp){
            if(state==State.canceled){
                return state;
            }else{
                hB=_hB;
                hBt=_hBt;
                state=State.finished;
            }
        }
    }
    function getState() returns (string){
        if(state==State.canceled) return "canceled";
        if(state==State.created) return "created";
        if(state==State.finished) return "finished";
    }
}
```

Listing 7: Confidential two-party certified notifications smart contract

variable *state*. The TTP will execute this function if it receives a request from the receiver. The smart contract is responsible for the publication in the blockchain of the values

of the elements used to maintain the fairness of the exchange following the protocol described in subsection 8.2.2. In function *Finish*, if the conditions are fulfilled, the smart contract publishes in the blockchain both the non-repudiation of reception proof (hB) and the session key encrypted with B's public key (hBt).

8.4 Security properties analysis

8.4.1 Non-confidential Certified Notifications without TTP

The non-confidential certified notifications protocol allows the fair exchange of a message and non-repudiation proofs. The main properties achieved by the protocol are analyzed in this section.

- **Strong Fairness.** *A* will not receive the non-repudiation proof of reception provided by the smart contract unless she executes the transaction to register the message in the blockchain (case *State=finished*). On the other hand, *B* will not have access to the message unless he executes the transaction to accept the notification (*State=Accepted*). While it is true that the information on the blockchain is public and anyone can execute the smart contract code locally, the protocol ensures that the smart contract does not generate alternative cancellation or finalization proofs that could create contradictory evidence. Additionally, potential race conditions where nodes could delay or prioritize transactions for their own benefit are mitigated by the inherent consensus mechanisms of the blockchain network, ensuring fairness and order of transactions.

- **Total absence of centralized TTP. Substitution by a decentralized smart contract.** This proposal does not require an external centralized party acting as a TTP. Instead, the blockchain itself, through smart contracts, fulfills this role in a decentralized manner. This decentralized TTP provides enhanced security and trust by removing the single point of failure and central authority. The parties execute the functions of the smart contract creating the associated transactions, and there is no need for dispute resolution by an external entity.

- **Transferability of the proofs.** Since the parties cannot obtain contradictory proofs in any way, the generated proofs can be presented as evidence to an external entity. Moreover, its transferability is easy, since the results of the exchange are stored in the blockchain. Due to the immutability of the blockchain, the content of the notification cannot be modified so the system provides integrity to the notification. The moment that the notification takes place can be derived from the timestamp of the block where the transaction is included.

- **Weak Timeliness.** The protocol is not asynchronous. If one of the parties delays its intervention in the exchange, the other party will not be able to resolve it until the deadline. However, after the deadline, both parties can request the finalization of the exchange. Moreover, the protocol wants to motivate the sender to conclude the exchange before the timeout blocks an amount of money in the smart contract. This amount will only be refunded to the sender if she concludes before the deadline.

- **Non Repudiation.** The protocol achieves non-repudiation of origin together with non-repudiation of receipt after the execution of the exchange. *A* cannot deny having sent the message since there is a transaction on the blockchain from her address containing the message and another one related to the same message including the address of the receiver and the hash of the message. *B* cannot deny having received the notification since there is a transaction from his address in the blockchain accepting the reception of the message and the *State* of the exchange is *Finished*, so the message is publicly accessible in the blockchain.

8.4.2 Confidential Certified Notifications with Stateless TTP

The main properties achieved by the confidential certified notifications protocol are analyzed in this section.

- **Weak Fairness.** The protocol does not allow any of the parties to receive the expected item if the other party does not receive it. However, the intervention of the TTP can lead to a situation in which one of the parties possesses contradictory evidence. A malicious *A* can have the non-repudiation proof received directly from *B* and also the cancellation proof generated by the smart contract after a cancellation request from *A*. For this reason, the fairness will be weak and the generated proofs are non-transferable. Comparing this feature with the version of the protocol without blockchain, this protocol does not require that the arbitrator consult both parties to resolve the final state of the exchange. It is enough to check one of the parties and then match this version with the contents of the blockchain.

- **Optimistic.** The parties can finalize the exchange without the need to contact a TTP or execute any function of the smart contract. If the parties do not follow the protocol and the execution of the smart contract is required, the gas necessary for its operation would be reduced compared with the protocol for non-confidential notifications.

- **Stateless TTP.** When the TTP is involved in the exchange, it can resolve the exchange through the use of the smart contract. The TTP does not need to store any kind of state information of the exchange.

- **Timeliness.** The parties can finish the exchange at any moment by accessing the smart contract (sender *A*) or contacting the TTP (receiver *B*). The duration of the resolution will depend on the block notification treatment. The protocol can assume that the transactions are valid immediately (zero confirmation) or wait until the block is confirmed in the chain (full confirmation).

- **Non repudiation.** The protocol achieves non-repudiation of origin together with non-repudiation of receipt after the execution of the three-step exchange or the finalization using the smart contract. *A* cannot deny having sent the message since *B* has the element received in the third step or the state of the smart contract is *Finished*. *B* cannot deny having received the notification since *A* has the elements sent by *B* in the second step of the protocol.

- **Confidentiality.** If the exchange is finished through the execution of the three-step exchange protocol, then no other entity is involved in the exchange, and the message remains confidential. If the TTP is involved or the functions of the smart contract are executed, then the TTP will process the received elements and will make a transaction including the element that will allow B to decrypt the message but the plain message is not included in the transaction so it will not be included in a block of the blockchain to preserve confidentiality.

8.5 Performance analysis

Table 8.1 and Table 8.2 present the gas required for the execution of each function, for both protocols.

The comparison between non-confidential and confidential smart contract deployments reveals distinct gas consumption patterns, which serve as a proxy for transaction costs and computational resources required. For the non-confidential version, deployment is significantly more resource-intensive, consuming 1,086,913 gas, compared to the confidential version, which requires 800,433 gas, indicating a more efficient initialization process for the latter. The operation costs vary between contexts: for instance, the 'Accept' action, specific to the non-confidential contract, costs 43,644 gas, whereas the confidential contract's comparable actions, such as 'Finish' and 'Cancel', are slightly different in nature and cost 88,387 and 44,698 gas respectively. Notably, the 'Finish' operation in the confidential setting shows a marked increase in gas requirement compared to the non-confidential setting.

Table 8.1: Gas cost of the non-confidential two-party certified notifications smart contract

	Non-Confidential Two-party Certified Notifications
Deployment	1086913
Accept	43644
Finish	59835
Cancel (created)	53011
Cancel (accepted)	30443

Table 8.2: Gas cost of the confidential two-party certified notifications smart contract

	Confidential Two-party Certified Notifications
Deployment	800433
Finish (Canceled)	26388
Finish	88387
Cancel	44698
Cancel (Finished)	24772

8.6 Conclusions

Previous solutions for fair certified notifications are mainly based on the intervention of a TTP that acts as an intermediary between sender and receiver. In this model of fair exchange, both parties obtain the expected item from the other or neither obtains what was expected. That is, either the issuer has received non-repudiation of reception evidence and the recipient has received the message, or neither party obtains the desired item, the TTP can intervene to guarantee the fairness of the exchange if some participant misbehaves.

This chapter presents two alternatives for sending certified notifications on a blockchain-based fairness. On the one hand, the first solution (see subsection 8.3.1) allows users to send non-confidential notifications, the new DApp supports the sending and receiving of certified messages and guarantees the fairness of the exchange without requiring the intervention of any TTP to guarantee the security properties of the exchange since the actions of the different actors are recorded in the blockchain and, in the event that any actor does not fulfill the protocol, the smart contract will generate the corresponding evidence to preserve fairness. This proposal also preserves the properties of limited Timeliness (involved parties can be certain that the protocol will be completed at a certain finite point in time[103]), Transferability of proofs and Non-repudiation as it is stated in subsection 8.4.1.

Table 8.3: Comparison of the properties of the non-confidential and confidential two-party certified notifications protocols

Property	**Non Confidential Notifications**	**Confidential Notifications**
Non-repudiation	YES	YES
Fairness	STRONG	WEAK
Timeliness	LIMITED	YES
Effectiveness	YES	YES
TTP	NO	OPTIMISTIC/STATELESS
Evidence Transferibility	YES	NO
Confidentiality	NO	YES

On the other hand, the second solution (see subsection 8.3.2) is a fair exchange protocol that enables the transmission of confidential notifications and introduces an optimistic TTP, whose intervention is only required if a party does not fulfill the protocol. Using blockchain technology and a smart contract allows the TTP to operate statelessly; it does not need to maintain the state of the exchange for any protocol execution as all transaction data is stored on the blockchain via the smart contract. This method ensures weak fairness (see subsection 8.4.2), maintaining Timeliness and Non-repudiation properties like the first solution. However, it does not strictly provide Transferability of proofs since verifying the correctness of the exchange requires not only examining the evidence provided by the parties but also consulting the blockchain (Table 8.3 compares the properties of both solutions). The choice of system, as presented in this chapter, largely depends on the specific needs of each exchange.

CHAPTER 9

MULTIPARTY CERTIFIED NOTIFICATIONS PROTOCOLS

In Chapter 8, we have presented two certified notifications protocols used for individual interactions. One focusing on non-confidential notifications and the other on confidential notifications. These protocols were specifically designed for communication between a single sender and a single receiver. The protocol for non-confidential notifications operates independently of a Trusted Third Party (TTP), whereas the confidential notification protocol requires one, to mediate and resolve disputes.

This chapter introduces two innovative protocols aimed at multiparty certified notifications, broadening the scope of our earlier efforts to include scenarios with multiple recipients. The first of these new protocols continues to handle non-confidential notifications and operates independently of a TTP, enhancing its utility by supporting multiple receivers simultaneously. The second protocol, which manages confidential notifications, still requires the involvement of a TTP for dispute mediation, thus ensuring secure and private communication across multiple parties. Together, these protocols mark a considerable evolution in the domain of secure communications, enhancing both autonomy and operational efficiency.

Central to our discussion is the typical case of certified notifications where the primary elements exchanged are the delivered data accompanied by Non-Repudiation of Origin (NRO) and Non-Repudiation of Reception (NRR) proofs. This framework is vital for understanding the operational dynamics of our newly introduced protocols. Both protocols use smart contracts on the Ethereum blockchain to ensure integrity and non-repudiation, emphasizing the decentralized nature of blockchain technology. We thoroughly analyze the properties of these protocols, such as security, and cost-efficiency, and provide a comparative performance analysis. This comprehensive examination not only clarifies the practical applications of our protocols but also sets the stage for future innovations in blockchain-based secure communication.

9.1 Contribution

This chapter significantly advances the field of secure multiparty communications by introducing two innovative protocols for multiparty certified notifications. Using blockchain technology, these protocols establish a secure and efficient framework for the delivery of notifications from a single sender to multiple receivers, reducing or even removing the role of the Trusted Third Partys (TTPs) inside such protocols.

The first protocol is tailored for non-confidential notifications, facilitating transparency and public verifiability by allowing the notification content to be accessible to all. This protocol is designed to function without a TTP, reflecting a major shift towards more decentralized notification systems. The second protocol ensures privacy and security for confidential notifications by keeping the notification content hidden from all except the intended receivers. Remarkably, this protocol is particularly notable for its minimal reliance on a TTP, which is only involved in dispute resolutions, making it an optimistic protocol.

Both protocols are designed to comply with the European Union norms for registered eDeliveries, as outlined in subsection 5.1.1. This compliance demonstrates not only their practical applicability but also their alignment with current regulatory frameworks. By offering solutions for both public and private communications, these protocols provide the versatility needed to cater to a wide range of applications. As noted in the directive that does not mandate confidentiality for all types of notifications [80], these protocols are structured to allow flexibility in the confidentiality of the delivered messages, attending to the specific needs of different use cases.

In addition to discussing the implementation details of the protocols using Ethereum smart contracts, this chapter highlights how smart contract technology can be effectively used to manage multiparty interactions. The analysis covers various aspects of the protocols, including security, cost-efficiency, and performance, providing a comprehensive overview of their capabilities and advantages.

This chapter analyzes the potential of blockchain-based technologies to provide two multiparty certified notification schemes that significantly reduce the reliance on Trusted Third Partys compared to traditional approaches while meeting the standards set by the European Union for registered eDeliveries. Notably, the protocols developed here do not require any TTP involvement for the non-confidential and confidential notifications.

Moreover, the protocols have been fully analyzed to ensure they satisfy both the legislative requirements and the ideal properties for such systems, like fairness in exchange. Performance analyses compare the costs of deploying and executing the main functions of the smart contracts, with findings suggesting superior efficiency over two-party systems. These protocols represent a pioneering effort in the realm of blockchain-based multiparty certified notifications, achieving key property, privacy, and performance goals.

9.2 Protocol design

This section presents an overview of the system, describing the participating actors, the two proposed alternatives, the methods and the interactions among the actors.

The proposed system presents solutions for both confidential and non-confidential (or public) certified delivery of messages. The proposals consider the following actors with these roles:

- Sender (A). The user that generates the data to deliver, chooses the receiver or set of receivers and sends the message. The sender also provides a Non-Repudiation of Origin proof.
- Receivers (B). The user or set of users that receive the delivery. The receivers must accept the delivery and provide a Non-Repudiation of Reception proof. The delivery can be accepted by a subset of receivers so the protocol must manage the exchange to maintain fairness in case of a selective acceptance.
- Trusted Third Party (TTP). Independent and trusted party that can act as an intermediary to solve disputes among senders and receivers. Only the confidential protocol involves a TTP.
- Smart Contract. Contract deployed on the blockchain that can manage, depending on the proposal, both the exchange of elements during the delivery and/or the resolution of disputes among senders and receivers.

All the participating actors (senders, receivers and TTP when it is required), possess blockchain addresses and are able to communicate with each other and with the smart contract. These entities interact as follows: first, the sender chooses whether he wants to send a confidential notification or a public notification. Then, in both protocols, the parties execute a three-step exchange to deliver the message and provide non-repudiation proofs:

- In the first step the sender proposes the message to be delivered and sends it in a hidden way to the set of receivers. This step must include elements to ensure the fairness of the exchange.
- Each receiver can choose individually if he accepts or not the delivery. The receivers who accept the delivery must send a Non-Repudiation of Reception proof related to the hidden message.
- The sender determines the group of receivers that has accepted the delivery and concludes the exchange by providing the message in clear and Non-Repudiation of Origin proof.

Both proposed protocols follow the three-step exchange. The main difference between the non-confidential and the confidential solution in regard to this three-step protocol is that in the confidential solution, the parties can exchange messages directly (*off-chain* communication exchange), while in the non-confidential solution, the parties execute the three-step protocol *on-chain*, by invoking functions of the smart contract deployed for this service. The *on-chain* interaction among the actors for the non-confidential protocol is depicted in Figure 9.1, while the *off-chain* interaction for the confidential protocol is depicted in Figure 9.2.

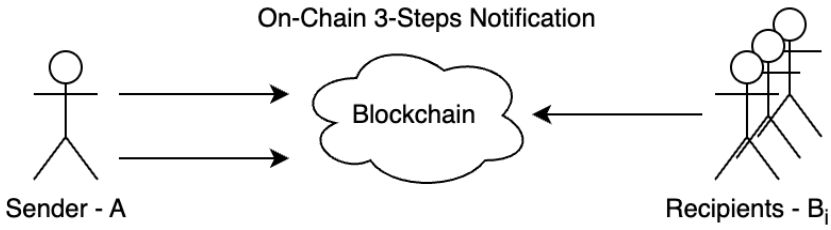


Figure 9.1: Interaction between the actors in the Non-Confidential Multiparty Ceritified Notifications protocol

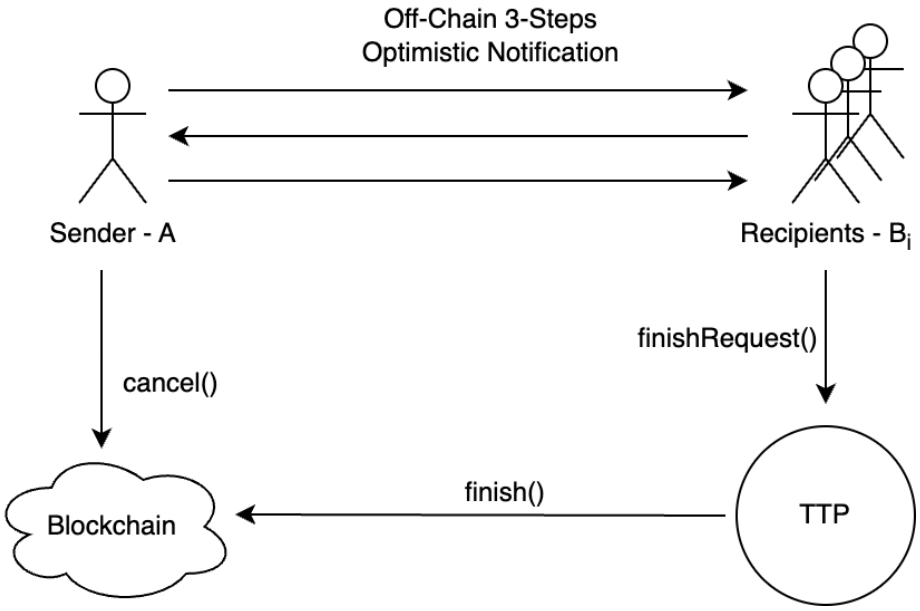


Figure 9.2: Interaction between the actors in the Confidential Multiparty Certified Notifications protocol

Due to the differences in the execution of the three-step protocol, a conflict resolution subprotocol has to be designed for the confidential solution in order to ensure fairness. This subprotocol uses the smart contract for the confidential certified notification service and involves a Trusted Third Party, as it is depicted in Figure 9.2.

Moreover, the three steps of the exchange are slightly different in the two protocols. They differ in the way they hide the data to deliver and also in the format of the non-repudiation proofs, as it will be explained in subsection 9.2.1 and subsection 9.2.2.

9.2.1 Non-Confidential Multiparty Certified Notifications

In Chapter 8 we presented a blockchain-based protocol for Certified Notifications for a single receiver. The proposal is a non-confidential fair exchange of a notification message together with a Non-Repudiation of Origin token for a Non-Repudiation of Reception token. This approach is well suited for those applications that require the storage of the delivered data, which must be registered and publicly accessible. How-

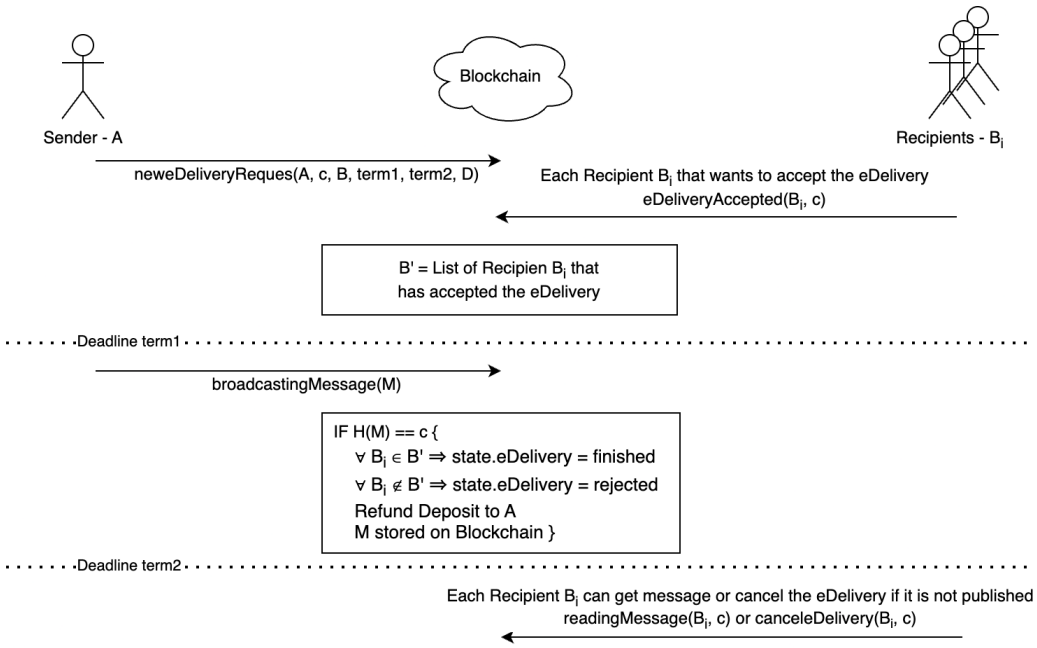


Figure 9.3: Description of the Non-Confidential Multiparty Certified Notifications

ever, the scheme is a two-party protocol, thus the notification can only have a single recipient.

Now, in this chapter, we present an improved non-confidential protocol with the new property of reusability and multiparty capabilities (i.e. multiple receivers), for certified notifications.

The multiparty protocol for non-confidential certified notifications presents a solution for fair deliveries with one Sender (A) and multiple Recipients ($B = \{B_1, B_2..., B_n\}$). Figure 9.3 is the diagram of the sequence of the messages exchanged by Sender and Recipients in the non-confidential certified notification for the on-chain communication scheme proposed. In Figure 9.3 the blue arrows represent the steps described above and designate the signed requests to a blockchain address. Also, the text in red inside the boxes describes the processes that have to be performed by the certified notifications service deployed on the blockchain.

The sender of the delivery, A, and the set of receivers, B, will follow the steps of the general exchange protocol depicted at the beginning of section 9.2. In the following complete description of the protocol, the requests sent by the actors of the protocol are directed to the address of the certified notifications service deployed on the blockchain. The details of the exchange protocol are (see Table 9.1 for notation):

1. The sender A sends a request to create a new notification. This request includes the identification address of the sender, the hash of the message to deliver (c, this hash code is also used as the identification number of the certified notification), the addresses of the receivers and the periods $term1$ and $term2$. $term1$ specifies the valid period for the receivers to accept the delivery before the sender finishes the delivery, and $term2$ specifies the time to allow receivers to cancel an unfinished delivery. We will see in section 9.4 that this cancellation is required

Table 9.1: Elements of the Non-Confidential Multiparty Certified Notifications Protocol

Elements	
A	Blockchain address of the sender A
B	Set of receivers
M	Message, data or file to deliver
B_i	Blockchain address of the receiver B_i
B'	Subset of B with the users that have accepted the delivery.
$term1$	Period for receivers to accept the delivery, specified as time since its creation.
$term2$	Period to allow receivers to cancel the delivery, specified as time since its creation.
$c = H(M)$	Collision-resistant hash function applied to M.
D	Deposit sent to the certified notifications Service.

to guarantee effectiveness and fairness. A deposit is used in this transaction to encourage finalization, but it could be optional.

2. Each receiver B_i in B has to individually accept the reception of the delivery during $term1$, publishing a message expressing his will. This signed transaction will act as a Non-Repudiation of Reception proof. If a receiver does not accept during $term1$, then a rejection is assumed.

3. After the deadline of $term1$, or after all receivers (members of B) have accepted the reception, sender A can publish the message by using the blockchain, finishing the delivery with the subset of receivers B' ($B' \subset B$) that has accepted the delivery. As a consequence, the certified notifications service deployed on the blockchain checks the integrity of the message and publishes the Non-Repudiation of Origin proof for the receivers in B'. After the finalization, the sender receives the refund of the deposit.

4. In this case, we have added a final step to the general three-step protocol: after the deadline of $term2$ any receiver in B' can get the message or can request the cancellation of the notification in case the message was not properly deposited in the previous step.

Finally, after the execution of the exchange protocol:

- All the receivers can read the message M since it is stored in the blockchain, but only members of B' can prove that they have been notified and have Non-Repudiation of Origin proofs.

- If after $term2$ the sender A hasn't published the message, each receiver B_i can cancel the notification. In this case, the state of these receivers will be $canceled$.

9.2.2 Confidential Multiparty Certified Notifications

The second proposal has been designed to take into account those deliveries that require confidentiality. That is, the blockchain has to help to preserve the fairness of the exchange but the message cannot be stored in a publicly accessible block.

The main difference with the non-confidential one is that the protocol, in the confidential case, allows an off-chain optimistic exchange, that is, the exchange can be executed completely without the intervention of the blockchain or the TTP.

Another important feature is that this proposal does not require a deadline and any exchange can be finished at any moment. A stateless TTP can be used to resolve the disputes that could arise between the parties.

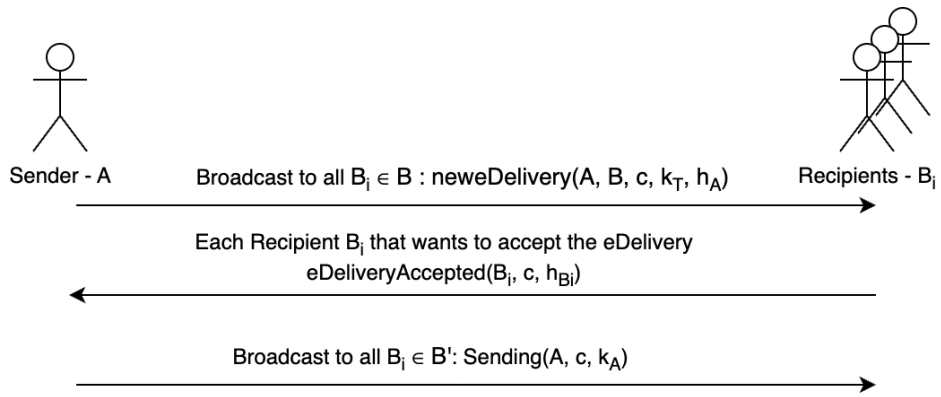


Figure 9.4: Optimistic Off-Chain Communication Subprotocol for the Confidential Multiparty Certified Notifications protocol

In [91, 95] we described a non-blockchain-based optimistic fair exchange that we partially reuse (i.e. the off-chain three-step exchange) and adapt for this purpose in the new blockchain-based proposal described in this chapter. In the new protocol, the originator *A* and the set of recipients *B* exchange messages and non-repudiation evidence directly, using the three-step off-chain communication depicted in subsection 9.2.2. Only as a last resort, in case they cannot get the expected items from the other party, the smart contract or the TTP would be invoked, by sending a *cancellation request* (Figure 9.5) or a *finish request* (Figure 9.6). In comparison with the protocol described in [95], the role of the TTP has been substantially reduced in the blockchain-based solution. Moreover, the sender will never contact the TTP. In this new proposal, the TTP will answer only requests from the receivers (members of *B*) by accessing the smart contract that has been deployed. The TTP is totally stateless, so it never stores information about the state of any exchange.

Our new multiparty protocol has three subprotocols. Next, we explain in detail these subprotocols, including the values that must be exchanged by the different parties. The subprotocols are: *Optimistic Exchange, Cancel* and *Finish* Subprotocols. Note that blue arrows in Figures represent signed request to a blockchain address, while black arrows are off-chain communication messages. Also, the text in red inside the boxes describes the processes that have to be performed by the certified notifications service deployed on the blockchain. Moreover, the description of the protocol follows the

Table 9.2: Elements of the Confidential Multiparty Certified Notifications Protocol

Elements	
M	Message, data or file to deliver.
A	Sender of the delivered data.
B	Set of Receivers.
B_i	A specific receiver.
B'	Subset of B with the users that will finish the exchange with A.
B''	Subset of B with the users that will not finish the optimistic exchange with A.
$B'' - finished$	Members of B that have contacted the TTP before the cancellation of the exchange.
$B'' - canceled$	Members of B that have not contacted the TTP before the cancellation of the exchange.
$c = E_K(M)$	Symmetric cipher of message M with key K.
$k_T = PU_T(K)$	Key K encrypted with TTP's public key
$h_A = PR_A[H[H(c), B, k_T]]$	A's signature (using her private key) on the concatenation of the hash of c, B and k_T and Id. First part of Non-Repudiation of Origin proof.
$h_{Bi} = PR_{Bi}[H[H(c), k_T]]$	B_i's signature (using his private key) on the concatenation of the hash of c and k_T and Id. Non-Repudiation of Reception proof.
$k_A = PR_A[K, B']$	A's signature (using her private key) on the key K together with B'. Second part of the Non-Repudiation of Origin proof for $B_i \epsilon B'$.
$k'_T = PR_T[K, B_i]$	TTP's signature (using its private key) on key k for user B_i. Alternative second part of the Non-Repudiation of Origin proof.
$h_{BiT} = PR_B[H[H(c), k_T, h_A, h_{Bi}]]$	Evidence of the TTP intervention requested by B_i.

notation included in Table 9.2.

Multiparty Optimistic Exchange Subprotocol.

The protocol is optimistic in the sense that it is possible for a sender A to complete the exchange with the set of receivers B without the intervention of the TTP. subsection 9.2.2 depicts the three-step off-chain message sequence exchanged by sender and recipients of a confidential notification to solve the exchange through the optimistic approach.

The exchange is as follows:

1. The sender sends a message to the set of receivers including the encrypted message, the addresses of the receivers and the first part of the Non-Repudiation of Origin proof, k_T, h_A.

2. Each receiver decides if he follows the exchange sending the Non-Repudiation of Reception proof, h_{Bi}.

3. The sender sends to each receiver, which has sent the message of step 2, a message containing the decryption key. Thus, this subset of receivers will receive the key to open the message and the remaining part of the Non-Repudiation of Origin proof, k_A.

If the execution of these steps has been successfully completed, the sender will hold Non-Repudiation of Reception (NRR) evidence from all recipients and every recipient

will hold the message and Non-Repudiation of Origin (NRO) evidence. Every recipient has the key and so he can decrypt the message, then each of the recipients of the set B obtains the key used to decipher the message, k_A, as well as the corresponding NRO evidence (h_A). In the same way, the sender of the message will obtain the NRR evidence (h_{Bi}) from each recipient. If some of the recipients don't send the message of step 2, those recipients won't receive the message of the last step. In fact, this last message contains the list of recipients that have completed the protocol. Therefore, a receiver that is not in the subset B' cannot use the message of step 3 received by other receivers as an NRO evidence. If some party does not follow this exchange protocol, the remaining users need to correct the unfair situation by requesting the cancel or finish resolutions.

This way, the protocol allows an optimistic exchange, that is, the exchange can be executed completely without the intervention of the TTP or the blockchain. Another important feature is that this proposal does not require a deadline and can be finished at any moment. The following subprotocols can be executed if disputes arise between the parties.

Multiparty Cancel Subprotocol.

The Cancel subprotocol will be initiated by the sender of the message. The sender executes the corresponding function of the smart contract in case of not receiving the element h_{Bi} from all the recipients of the message. In Figure 9.5 there is a graphical description of the actions taken by the sender and the blockchain to cancel a confidential notification for non-finished recipients. The hash code $H(c)$ is used as the identification number of the certified notification.

When the sender executes the *Cancel* function of the smart contract, she has to indicate the identity of all those users who have not sent h_{Bi} (represented by the set B"). The smart contract, for its part, will be responsible for checking if any of the users in the set B" has already finished the exchange employing the TTP. In this case, it will

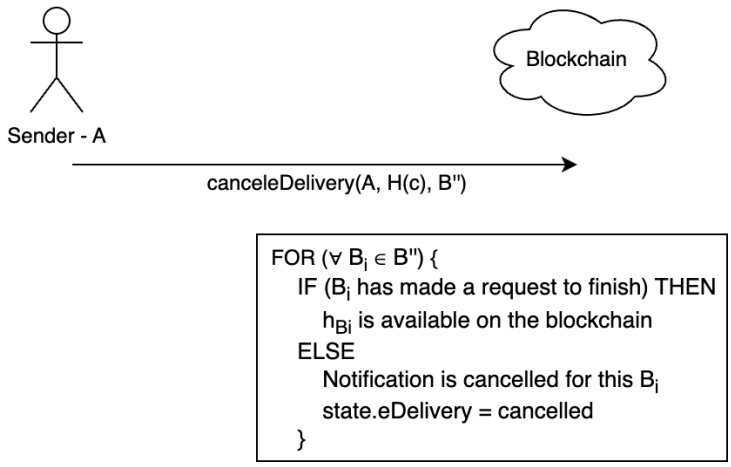


Figure 9.5: On-Chain Cancel Subprotocol for the Confidential Multiparty Certified Notifications protocol

send the corresponding NRR evidence (for a particular B_i) to the sender. Otherwise, the unfinished recipients will be included in the group of canceled users ($B" - canceled$). Therefore, at the end of this phase, the sender A will have concluded the fair exchange with all recipients, either satisfactorily, because she has received the correspondent h_{Bi}, or as a result of a cancellation.

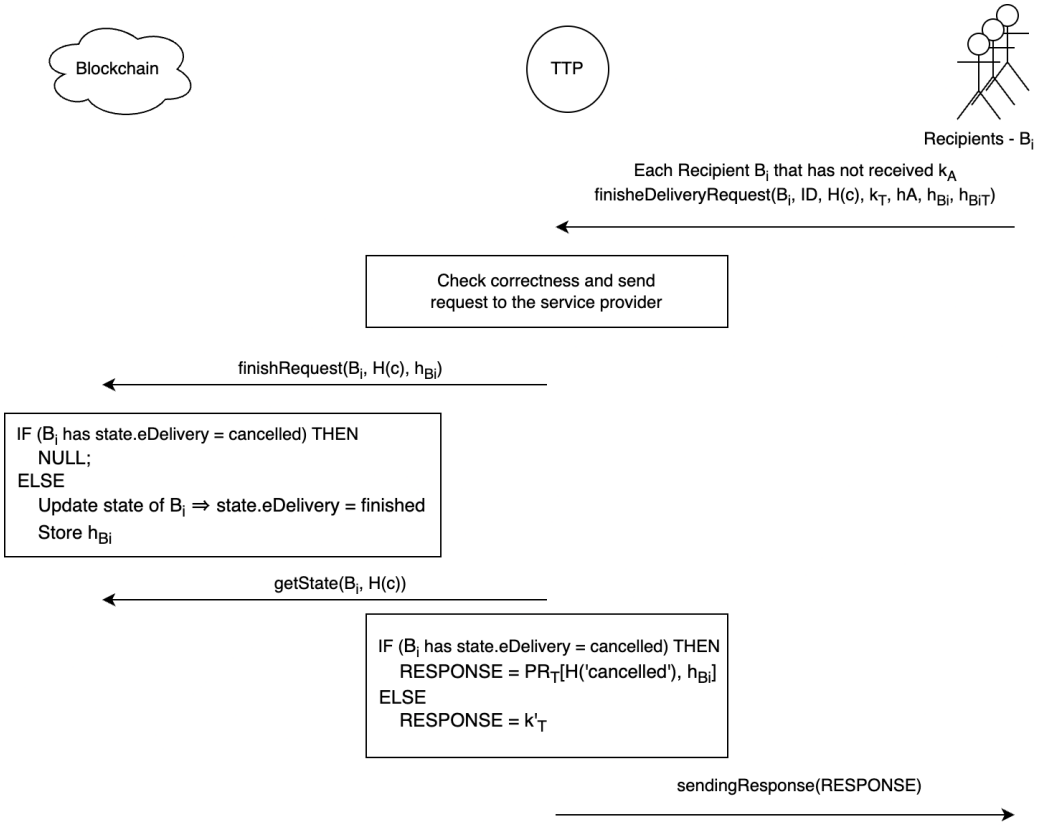


Figure 9.6: On-Chain Finish Subprotocol for the Confidential Multiparty Certified Notifications protocol

Multiparty Finish Subprotocol.

The Finish subprotocol will be initiated by any receiver, in the case of having sent the corresponding h_{Bi} but not having received the element to obtain the encryption key, k_A. This finalization will be carried out by the TTP based on the request received from a recipient B_i. After checking the correctness of all the different parameters received from B_i the TTP executes the *finish* function of the smart contract (the TTP submits B_i's NRR evidence (h_{Bi}) to the smart contract). In Figure 9.6 there is the description of the actions taken by recipients, TTP and blockchain to finish a confidential notification in case of exception. The TTP's response is based on the information stored on the blockchain about this certified notification.

In this subprotocol, the smart contract checks that the request comes from the TTP, and then it verifies if the claimed recipient is among the users whose message

delivery has been canceled. In this case, the appropriate cancellation evidence is issued. Otherwise, the smart contract stores in the blockchain the received h_{Bi} and updates the set of users who have finished this exchange by adding B_i to $B" - finished$. Finally, the TTP sends k_T to B_i in order to enable the recipient to read the message and to complete the NRO evidence.

9.3 Implementation

The implemented protocols depicted in subsection 9.2.1 and subsection 9.2.2 can be accessed via the *NonConfidentialMultipartyRegisteredEDelivery*[1] and *Confidential-MultipartyRegisteredEDelivery*[2] GitHub repositories maintained by the Security and e-Commerce (SECOM) Research Group from the University of the Balearic Islands.

9.3.1 Non-Confidential Multiparty Certified Notifications

In this protocol, we have used a *Smart Contract* to allow the sender A to send the same non-confidential message to several receivers, members of B, exchanging non-repudiation evidence to complete the delivery.

Data structures are required to store the set of receivers and provide multiparty capabilities, so an array stores the list of addresses of the receivers, and a mapping stores the state of the exchange for each receiver. As it has been explained in subsection 9.2.1, in this multipaty protocol the state of the exchange depends on each receiver. We use a mapping to get a constant time search, and, at the same time, a constant cost search. But mappings are not iterable. For this reason, we have also an array to store all addresses of the receivers, to iterate through all these addresses.

The contract also needs to save the *messageHash* and the *message* values of the certified notification and use two periods: *term1* and *term2*. The *start* variable stores the time when the delivery is created, and is used to set the timeouts. *acceptedReceivers* will count the number of receivers that have accepted, to let the sender finish the delivery,

[1] https://github.com/secomuib/NonConfidentialCertNotification/tree/master/contracts
[2] https://github.com/secomuib/ConfidentialCertNotification/tree/master/contracts

```
enum State {notexists, created, canceled, accepted, finished,
    rejected}

address public sender;
address[] public receivers;
mapping (address => State) public receiversState;
uint acceptedReceivers;

bytes32 public messageHash;
string public message;

uint public term1;
uint public term2;
uint public start;
```

Listing 8: Variables of the Non-Confidential smart contract

```
function accept() public {
  require(now < start+term1, "The timeout term1 has been reached");
  require(receiversState[msg.sender]==State.created, "Only receivers
      with 'created' state can accept");

  acceptedReceivers = acceptedReceivers+1;
  receiversState[msg.sender] = State.accepted;
}
```

Listing 9: accept() function

```
  function finish(string _message) public {
    require((now >= start+term1) || (acceptedReceivers>=receivers.
        length), "The timeout term1 has not been reached and not all
         receivers have been accepted the delivery");
    require (msg.sender==sender, "Only sender of the notification
        can finish");
    require (messageHash==keccak256(_message), "Message not valid (
        different hash)");

    message = _message;
    sender.transfer(this.balance);
    for (uint i = 0; i<receivers.length; i++) {
      if (receiversState[receivers[i]] == State.accepted) {
        receiversState[receivers[i]] = State.finished;
      } else if (receiversState[receivers[i]] == State.created) {
        receiversState[receivers[i]] = State.rejected;
      }
    }
  }
```

Listing 10: finish() function

avoiding waiting until the deadline of *term1*. The use of this variable will avoid the need to check the state of all receivers, which is expensive in terms of gas. These data structures can be seen in Listing 8.

When a new *Smart Contract* of this type is created, an array of receivers are set with the parameters passed by the sender, and then the state of every receiver of the delivery are initialized to *created*.

The *Accept* function (see Listing 9) checks that the sender of the function is in the receivers mapping, and its delivery state is *created*. It also checks that the deadline of *term1* has not been reached. If all these conditions are satisfied, the delivery state of this receiver is set to *accepted*.

The *Finish* function (see Listing 10) checks that the deadline of *term1* has been reached or if all receivers have been accepted. The function also verifies that its caller is the sender of the notification, and that the text of the message matches the hash specified before. If all of these conditions are fulfilled, the delivery states of the receivers that were *accepted* are set to *finished*, and the states of the receivers that were *created* are set to *rejected*.

Finally, the *Cancel* function (see Listing 11) checks that the *term2* timeout has been

```
function cancel() public {
  require(now >= start+term2, "The timeout term2 has not been
      reached");
  require(receiversState[msg.sender]==State.accepted, "Only
      receivers with 'accepted' state can cancel");

  receiversState[msg.sender] = State.canceled;
}
```

Listing 11: cancel() function

reached and that the function is called by a receiver with delivery state *accepted*. If all these conditions are satisfied, the receiver's delivery state is set to *canceled*.

In order to create and deploy reusable certified notification contracts we use a commonly used design pattern, a *Factory Contract*, a type of contract that creates and deploys other contracts. With this pattern, we achieve the reusability of the Non-Confidential Multiparty Certified Notification contracts, and we get the following advantages:

- The factory can be used to store the addresses of the child contracts so that they can be extracted whenever necessary. This is safer than storing it in an external database, preventing the loss of references to these contracts.
- Our front-end service only needs to store the address of the factory contract.
- The front-end service has to pay only for the deployment of this factory contract.
- Users could invoke a function in the factory to deploy a new delivery contract, paying deployment costs (see section 9.5). Then, the factory itself deploys the delivery contract and stores the address of the deployed contract in an internal list.

9.3.2 Non-Confidential Multiparty Certified Notifications

In the Confidential Multiparty blockchain-based protocol, the sender A and the set of receivers B will exchange messages and non-repudiation evidence directly. Only as a last resort, in the case they cannot get the expected items from the other party, the smart contract or the TTP would be invoked, by calling their *cancel* or *finish* functions.

A *Smart Contract* based on the smart contract presented in Chapter 8 is used, adapted to the new protocol, so now there is a possibility for a message to be sent to several receivers. A data structure allows the association of multiple receivers to the same message and is in charge of storing the necessary parameters and determining the state of the exchange. The state of the exchange is no longer uniquely defined in the message but depends on each receiver. The set of receivers will be stored within the *struct* message by using a *mapping*, as can be seen in the code in Listing 12. In the *ttp* variable, we will store the address of the account that creates this smart contract.

The function *Cancel* (Listing 13) uses as the input parameter an array that contains the set B". The function goes through this set and cancels the exchange for each receiver if it does not exist previously.

```
enum State {notexists, canceled, finished }

struct ReceiverState{
  bytes32 receiverSignature;
  bytes32 keySignature;
  State state;
}

struct Message{
  bool messageExists;
  address sender;
  address[] receivers;
  mapping(address => ReceiverState) receiversState;
}

mapping(address => mapping(uint => Message)) messages;
address public ttp;
```

Listing 12: Variables and constructor of the Non-Confidential smart contract

```
function cancel(uint _id, address[] _receivers) public {
  for (uint i = 0; i<_receivers.length;i++){
    address receiverToCancel = _receivers[i];
    if (messages[msg.sender][_id].receiversState[receiverToCancel
        ].state==State.notexists) {
      addReceiver(_id, msg.sender, receiverToCancel, 0, 0, State.
          canceled);
    }
  }
}
```

Listing 13: cancel() function

```
function finish(uint _id, address _sender, address _receiver,
    bytes32 _receiverSignature, bytes32 _keySignature) public {
  require (msg.sender==ttp, "Only TTP can finish");
  if (messages[_sender][_id].receiversState[_receiver].state==
      State.notexists) {
    addReceiver(_id, _sender, _receiver, _receiverSignature,
        _keySignature, State.finished);
  }
}
```

Listing 14: finish() function

Function *Finish* (Listing 14) checks if B_i exists. If B_i does not exist, the TTP includes it in *B"-finished*, and the NRR proof h_{Bi} and the key k, with its corresponding private key, are stored.

The smart contract also includes a function for the addition of a new receiver (Listing 15), that also checks if that message exists and creates it if necessary.

It is important to stress that unlike the non-confidential protocol, in the confidential protocol we don't use the *Factory Contract* pattern to create new deliveries. This is due

```
function addReceiver(uint _id, address _sender, address
    _receiver, bytes32 _receiverSignature, bytes32 _keySignature
    , State _state) private {
  if (!messages[msg.sender][_id].messageExists) {
    messages[_sender][_id].sender = _sender;
    messages[_sender][_id].messageExists = true;
  }
  messages[_sender][_id].receivers.push(_receiver);
  messages[_sender][_id].receiversState[_receiver].
      receiverSignature = _receiverSignature;
  messages[_sender][_id].receiversState[_receiver].keySignature
      = _keySignature;
  messages[_sender][_id].receiversState[_receiver].state =
      _state;
}
```

Listing 15: addReceiver() function

to the fact that in this protocol we only need to store simple variables like addresses, states and binary signatures, in contrast to the non-confidential protocol, where we need to use more complex structures and functions for each delivery. For this reason, we can store all this in a single smart contract, using mappings and structs, which is cheaper than deploying a smart contract for every single delivery.

9.4 Security properties analysis

This section presents an analysis of the ideal properties of a certified notification system (listed in section 5.1) for both proposals. The basis of this rationale about the security and privacy properties is the correct use of the cryptographic primitives. Our proposals make use of the following primitives:

- Digital Signatures
- Symmetric Encryption
- Public Key Encryption
- Key Wrapping
- Hash Functions

Thus, in order to make a secure implementation, any deployment has to take into consideration the official documents that address the use of cryptographic algorithms and which key lengths are specified. The last issue of the NIST document [169] includes an explanation of the projected maximum-security strength of key lengths associated with the cryptographic algorithms. Also, the document has a prediction of the period of time throughout the algorithms and the proposed key lengths are expected to provide probable security. That is to say, in order to break the security any attacker must solve the underlying problem of the implemented cryptographic operations. Thus, the protocols provide suitable and practice-oriented provable security as far as implementors

choose sufficiently large values of the security parameters and key lengths according to the international standards [169]. Regarding the specific cryptographic operations used in our protocols, the digital signature algorithms [170], key wrapping and public key encryption operations are based on the length and the proper generation of the domain parameters used.

In our proposals, we have both operations that are performed off-chain and on-chain. For the off-chain operations, in order to provide *acceptable* security (i.e. no security risk is currently known when used in accordance with any associated guidance), the DSA domain parameter lengths have to be (2048, 224) or (2048, 256), which provide a security strength of 112 bits; or (3072, 256), which provides a security strength of 128 bits [169]. Finally, hash function family SHA-2 specified in [171] provides *acceptable* security for all hash function applications. Regarding on-chain communication, signatures are used to authorize transactions on behalf of the signer. They are also used to prove to a smart contract that a certain account approved a certain message. Therefore, these signed messages can be used as non-repudiation evidence. Well-known blockchains such as Bitcoin and Ethereum apply the ECDSA algorithm to create signatures on transactions. In particular, Ethereum signatures use ECDSA and secp256k1 constants to define the elliptic curve and create secure signatures[172]. Providing that developers follow the standard guidelines to use this cryptography in the implementation of our protocol, the following discussion holds.

The discussion about security includes six propositions to evaluate properties: *effectiveness*, *fairness and evidence transferability*, *temporal parameters* (timeliness and timestamping), *non-repudiation*, *Trusted Third Partys* (presence, verifiability and maintenance of state information) and *confidentiality*. Each proposition is formed by different claims with its respective proofs and a final result.

The property of *efficiency* is not included in this analysis because we have performed a list of experiments to evaluate the efficiency of the protocols, and their results are included in section 9.5.

Proposition 1 ***Effectiveness.***
The proposed protocols for certified notifications are effective, that is, if the parties behave correctly, they will receive the expected items.

Claim 1 *The* non-confidential *protocol for multiparty certified notifications is* effective.

Proof: In order to send a new notification, the sender creates a new instance of the smart contract to perform the specified functions according to the regular operational mode of the specified protocol. If all parties invoke all the functions correctly, all of them will receive the expected items, as it is easily deduced from the smart contract solidity code included in Listing 8, Listing 9, Listing 10 and Listing 11 and in more detail in our GitHub repository. Actually, no TTP will be involved in the protocol in any case, even if any party does not follow the three-step exchange protocol, the smart contract will ensure a fair result for each party :

- If the receiver does not invoke the *accept()* function, then the sender can invoke the *finish()* function to solve the exchange.

- If the sender does not invoke the *finish()* function, then the receiver can invoke the *cancel()* function to solve the exchange.

Claim 2 *The* confidential *protocol for multiparty certified notifications is* effective.

Proof: This protocol has an optimistic approach for a notification issuing, thus if the parties correctly execute the off-chain steps specified in section 9.2.2 the exchange will successfully conclude without any intervention of the TTP and will produce the desired result for each one. Only if the sender does not execute step number 3 of this subprotocol, the receiver has to invoke the TTP to conclude the exchange. The first step uses public key encryption while all the steps include digital signatures. All these operations take into account the secure use of cryptographic primitives included in the beginning of this section.

Result 1 *According to what is said in Claim 1 and Claim 2, the proposed certified notifications protocols fulfill the property of effectiveness.*

Proposition 2 ***Fairness and evidence.***
The proposed protocols for certified notifications are fair, so after completion of a protocol run, either each party receives the expected item or neither party receives any useful information about the other's item. Moreover, in the proposed protocol for non-confidential certified notifications, the proofs generated by the system can be transferred to external entities to prove the result of the exchange.

Claim 3 *The multiparty* non-confidential *certified notifications protocol provides* strong fairness.

Proof: On the one hand, according to the protocol described in subsection 9.2.1, the sender A will not receive the Non-Repudiation of Reception proof provided by the smart contract unless she makes the transaction that registers the message on the blockchain (case *state.eDelivery=finished*). On the other hand, a recipient B_i will not have access to the message unless he executes a transaction to accept the certified notification (*state.eDelivery=accepted*). At any moment, the smart contract does not generate alternative cancellation or finalization proofs that could create any situation where one of the parties could have contradictory proofs (thus, there is no action that can lead the exchange to *weak fairness* situation), as can be seen in Figure 9.7.

Claim 4 *The generated proofs in the multiparty* non-confidential *certified notifications protocol can be presented as* evidence *to an external entity.*

Proof: Since the parties cannot obtain contradictory proofs in any way (evidence can be only generated by the logic of the smart contract), the generated proofs can be presented as evidence to an external entity. Moreover, its transferability is easy, since the results of the exchange are stored on the blockchain. Due to the immutability of the blockchain, the content of the message cannot be modified so the system provides integrity to the message.

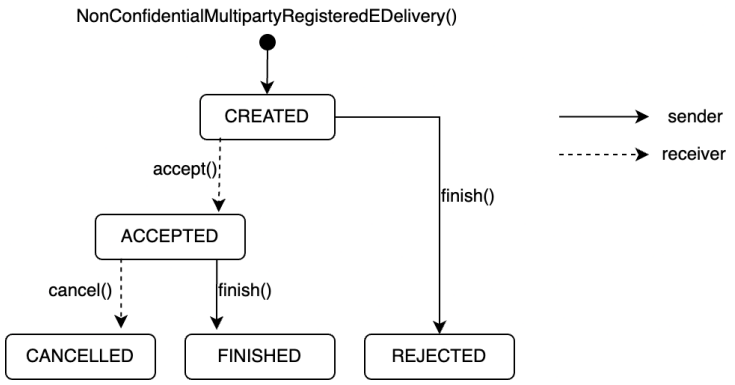


Figure 9.7: States in the Non-Confidential Multiparty Certified Notifications Protocol

The moment the delivery takes place can be derived from the timestamp of the block where the transaction was included.

Claim 5 *The multiparty* confidential *certified notifications protocol provides* weak fairness.

Proof: The confidential protocol, described in subsection 9.2.2, does not allow any party to receive the expected item from the other party unless the latter is in a position to get access to the correspondent expected item. However, the intervention of the TTP can lead to a situation where one of the parties possesses contradictory evidence. For instance, a malicious sender *A* can have the non-repudiation proof received directly from the recipient *B* and, in addition, she can get the cancellation proof generated by the smart contract after invoking the correspondent cancellation request to the smart contract. For this reason, the fairness will be weak and the generated proofs are non-transferable.

Claim 6 *The* confidential *certified notifications protocol proposed in subsection 9.2.2 does not have the property of transferability although the* evidence *of the final state of the exchange can be consulted in the blockchain.*

Proof: The sender can get a non-repudiation proof from the recipient in step 2 of the exchange protocol (section 9.2.2). However, this is not enough to certainly prove to a third party that the recipient has received the message. Additionally, the third party should check the information stored on the blockchain related to this particular notification to confirm this issue. Thus, this protocol does not provide transferable evidence, since a protocol generates transferable evidence only if the sender and recipient can separately demonstrate to any third party the result of the exchange without the need to request other entities.

Result 2 *In the non-confidential protocol, the evidence can only be generated by the smart contract. Thus, according to what is stated in Claim 3 and Claim 4, the smart*

contract will guarantee the fairness of the exchange (strong fairness) and provide transferability of evidence. In contrast, in the confidential protocol, actors can get evidence from the exchange protocol and from the smart contract. Thus, if a claim arises, the arbitrator has to consult both parties to resolve the final state of the exchange (weak fairness).

Proposition 3 ***Temporal parameters.***
The multiparty certified notification approaches in subsection 9.2.1 and subsection 9.2.2 offer timestamp and timeliness.

Claim 7 *The multiparty* non-confidential *certified notifications protocol satisfies* weak timeliness.

Proof: The protocol is not asynchronous. If one of the parties delays its intervention in the exchange, the other party will not be able to resolve it until the deadline. However, after the deadline, both parties can request the finalization of the exchange (see Figure 9.3, Listing 10 and Listing 11). Moreover, the protocol wants to motivate the sender to conclude the exchange before the timeout blocking an amount of money in the smart contract. This amount will only be refunded to the sender if she concludes before the deadline. All the transactions performed on the blockchain are timestamped.

Claim 8 *The multiparty* confidential *certified notifications protocol fulfills the* strong timeliness *property.*

Proof: The parties can finish the exchange at any moment by accessing the smart contract (sender A) or contacting the TTP (receiver B). The duration of the resolution will depend on the block notification treatment. The protocol can assume that transactions are valid immediately (zero confirmation) or wait until the block is confirmed in the chain (full confirmation). All the transactions performed on the blockchain are timestamped.

Result 3 *The confidential multiparty certified notifications protocol fulfills the strong timeliness property since either party can invoke the correspondent finalization procedure at any moment. However, the non-confidential protocol satisfies weak timeliness since parties cannot decide to finish the exchange sooner than the specific timeouts.*

Proposition 4 ***Non-repudiation.***
In the proposed protocols for certified notifications any sender cannot deny being the origin of an item and any recipient cannot deny being the receiver of an item either.

Claim 9 *The multiparty* non-confidential *certified notifications protocol achieves* Non-Repudiation of Origin *together with* Non-Repudiation of Reception *after the execution of the exchange.*

Proof: Regarding the sender A, she cannot deny having sent the message since there is a transaction on the blockchain from her address containing the message and another one related to the same message including the address

of the receiver and the hash of the message. With respect to the recipient B, he cannot deny having received the delivered data since there is a transaction from his address in the blockchain accepting the reception of the message and the *State* of the exchange is *Finished*, so the message is publicly accessible in the blockchain.

Claim 10 *The multiparty* confidential *certified notifications protocol provides* Non-Repudiation of Origin *together with* Non-Repudiation of Reception *evidence to the involved parties in an exchange.*

Proof: The protocol achieves Non-Repudiation of Origin together with Non-Repudiation of Reception after successful execution of the three-step exchange where secure cryptographic primitives are used or, if this subprotocol does not finish successfully, then users can complete the exchange by means of the smart contract. Thus, sender A cannot deny having sent the message since recipient B has the element received in the third step of the protocol (signed by A) or the state of the smart contract is *Finished*. In addition to that, B cannot deny having received the message since A has the elements sent by B in the second step of the protocol.

Result 4 *As it is stated in 9 and 10, senders and recipients will get the correspondent non-repudiation evidence from the proposed certified notifications protocols. Thus, certified notification schemes proposed in this chapter achieve the non-repudiation property.*

Proposition 5 ***Trusted Third Parties.***
The protocols proposed in this chapter are verifiable. A TTP is only used in the confidential multiparty protocol. The involvement of the third party in this protocol can be verified and it is not required that the TTP maintains state information.

Claim 11 *There is a total absence of* TTP *in the multiparty* non-confidential *certified notifications protocol. It has been substituted completely by the smart contract, but even so, the actions performed in the protocol are verifiable.*

Proof: This proposal does not require an external party acting as a TTP. Parties execute the functions of the smart contract creating the associate transactions and there is no need for dispute resolution. All the communications in this protocol are on-chain, thus, they are stored on the blockchain. Blockchain has been designed to be immutable and publicly verifiable, therefore the actions completed in the protocol can be verified and anyone can know which address is accountable for that.

Claim 12 *In the multiparty* confidential *certified notifications protocol, the* TTP *is only involved in case of exception. This third party is an optimistic stateless* TTP. *Moreover, the blockchain and the smart contract provide evidence of the* TTP *intervention.*

Proof: The TTP is not involved in the exchange subprotocol described in subsection 9.2.2. The TTP only intervenes in the protocol if the exception case: is when the recipient claims that she has not received the expected item from the sender, according to the first step of the exchange protocol. When the TTP is involved in the dispute resolution, it can resolve the exchange through the use of the smart contract, executing the *finish* function. The TTP does not need to store any kind of state information of the exchange.

The TTP is only able to invoke the *finish()* function of the smart contract according to the data provided by the recipient of the certified notification (the recipient will provide the *sender address* and the identifier to identify a notification). Then, the TTP will answer the recipient's request as the smart contract stated. If the TTP misbehaves, anyone who knows the notification identifiers (i.e. *sender address* and message identifier) can verify the answer of the TTP according to the data stored on the blockchain.

Result 5 *Thanks to blockchain technology there is no need for TTP involvement in the non-confidential certified notifications protocol. Nevertheless, the protocol is verifiable. In the confidential multiparty protocol, the TTP will only be involved if the recipient claims that she has not received the appropriate item from the sender. In this case, the TTP will act according to what is established in the smart contract, otherwise, the blockchain can provide evidence of any wrong behavior. Thus, the actions of the TTP are verifiable. Also, the blockchain allows keeping public and verifiable the state of any certified notification that uses the confidential scheme, therefore the TTP involved in the exchange is stateless.*

Proposition 6 ***Confidentiality.***
Confidentiality is an optional property in certified notifications protocols. The protocols proposed in this chapter provide solutions for both confidential and non-confidential (or public) deliveries.

Claim 13 *Since* confidentiality *is an optional property, the multiparty* non-confidential *certified notifications protocol does not implement this feature, allowing public deliveries.*

Proof: Instead of keeping the message secret, the protocol specified in subsection 9.2.1 stores the data related to any exchange on the blockchain. Thus, it offers the possibility to access the message of any certified notification through a blockchain explorer.

Claim 14 *The multiparty* confidential *certified notifications protocol ensures the* confidentiality *of the delivered data.*

Proof: If the exchange is finished through the execution of the three-step exchange subprotocol, then no other entity is involved in the exchange, and the message remains confidential (the message is encrypted using a symmetric cipher prior to sending it). If the TTP is involved or the functions of the smart contract are executed, then the TTP will process the received elements and will

make a transaction including the element that will allow the recipient B to decrypt the message but the plain message is not included in the transaction so it will not be included in a block of the blockchain to preserve the confidentiality.

Result 6 *The confidential blockchain-based protocol of subsection 9.2.2 can be used if the confidentially of the message is desired in a certified notification exchange. Otherwise, when the message has to be public, the scheme of subsection 9.2.1 is suitable.*

9.5 Performance analysis

This performance analysis includes experiments to determine the efficiency of the system in terms of cost since the economical execution costs could be a concert in the development of the certified notifications service. These tests have been performed using the *Smart Contracts* explained in section 9.3, deployed in the Ganache network, a personal blockchain used for Ethereum development, to isolate the performance conditions and possible problems of a real network like the main Ethereum network or the Rinkeby test network. For both Non-Confidential and Confidential protocols, we have detailed the gas cost of their main functions, comparing the Two-Party version with the Multiparty protocol for one, two and ten receivers. With these tests, we have two main objectives: on the one hand we want to obtain absolute values of the economic costs to evaluate the viability of the protocols and on the other hand we want to evaluate if the multiparty protocols are able to reduce the cost of the Two-party protocols.

We have also tested the performance in terms of delay, to have an illustrative reference of the delays introduced by the functions, although we have not included the complete list of results in this chapter. In the Ganache network, all the functions have been executed with a delay between 35 and 170 milliseconds, closely related to the gas cost of each function. However, in a normal production Ethereum blockchain network, the transaction validation delay varies due to the recent transition to Proof of Stake (PoS), generally taking between 15 to 30 seconds. Additionally, the delay depends on the transaction fee the sender is willing to pay, as higher fees can result in faster inclusion in the blockchain.

Regarding the protocols, we can say that they use the minimum number of steps that allow the effective exchange. In the confidential protocol, moreover, the parties can finalize the exchange without the need to contact a TTP or execute any function of the smart contract. If the parties do not follow the protocol and the execution of the smart contract is required, the gas necessary for its operation would be reduced compared with the protocol for non-confidential certified notifications protocol.

The exact value of the execution cost will be useful to check when a multiparty version of a protocol would be more efficient than a two-party protocol. Moreover, the results will be also useful to compare the cost of the non-confidential and the confidential protocols.

In Figure 9.8 we can see the gas cost of the main functions of the Non-Confidential two-party and multiparty protocol. From their analysis we can conclude:

- The deployment of the factory and the deployment of the delivery (*createDelivery()* function) are considerably more expensive than the *accept()* and *finish()* functions.

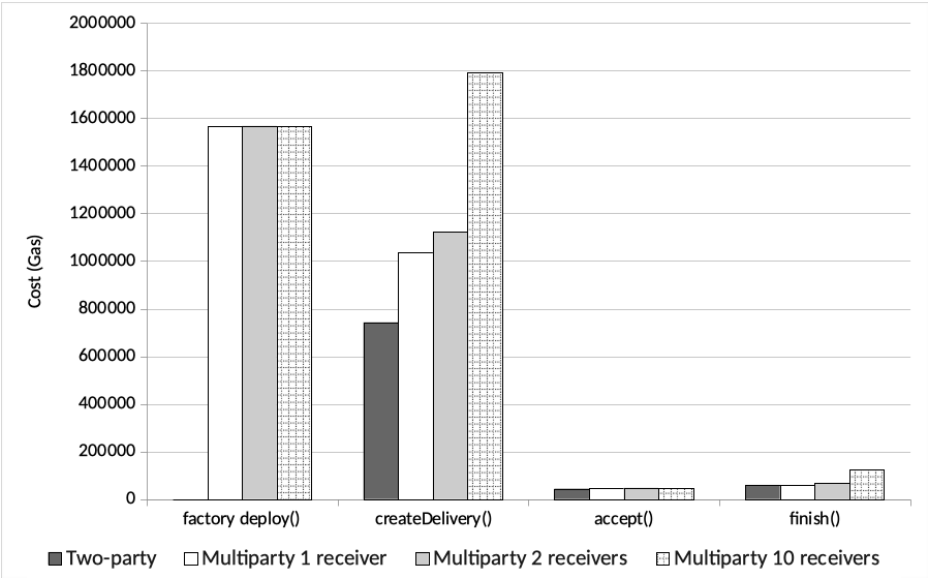


Figure 9.8: Gas cost of the Non-Confidential multiparty certified notifications smart contract

Table 9.3: Non Confidential multiparty certified notifications protocol cost in gas and equivalent US-Dollars price with 1 Gwei - 20 Gwei gas price

Cost in Gas	**Two-party**	**Multiparty 1 receiver**	**Multiparty 2 receivers**	**Multiparty 10 receivers**
factory deploy()	0	1,568,284 (0.23-4.58)	1,568,284 (0.23-4.58)	1,568,284 (0.23-4.58)
create-Delivery()	742,569 (0.11-2.17)	1,038,402 (0.15-3.03)	1,122,305 (0.16-3.28)	1,793,522 (0.26-5.23)
accept()	43,545 (0.01-0.13)	47,711 (0.01-0.14)	47,711 (0.01-0.14)	47,711 (0.01-0.14)
finish()	59,489 (0.01-0.17)	60,642 (0.01-0.18)	67,725 (0.01-0.20)	124,389 (0.02-0.36)

- We have to take into account that the cost of the two more expensive operations (deploy and *createDelivery* function) is distributed between the owner of the service (the *Factory* deployer) and the sender of the delivery.

- It is cheaper to deploy a two-party delivery than a Muliparty delivery with only one receiver. But when the same data have to be sent to two or more receivers, it is cheaper to do it with the Multiparty protocol, because it avoids the deployment of one smart contract for each receiver.

In Figure 9.9 we can see the performance in terms of gas cost of the Confidential two-party and multiparty protocols. We can highlight the following conclusions:

- The gas cost is, in general, cheaper than the Non-Confidential protocol, but we have to consider that the smart contract of the Confidential protocol will be used only when the TTP is required. The rest of this protocol's functionalities are out of the blockchain.

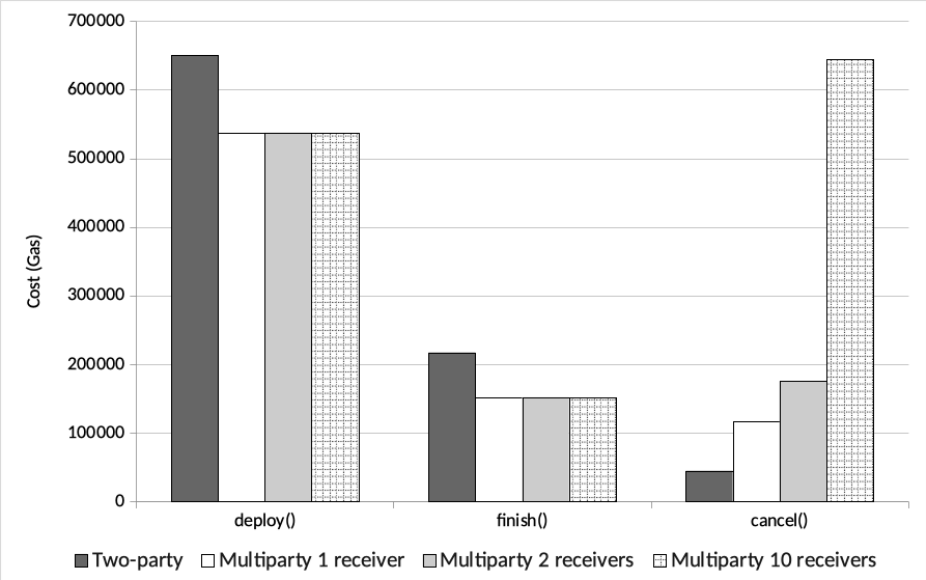


Figure 9.9: Gas cost of the Confidential multiparty certified notifications smart contract

Table 9.4: Confidential multiparty certified notifications protocol cost in gas and equivalent US-Dollars price with 1 Gwei - 20 Gwei gas price

Cost in Gas	**Two-party**	**Multiparty 1 receiver**	**Multiparty 2 receivers**	**Multiparty 10 receivers**
deploy()	650,451 (0.09-1.90)	537,397 (0.08-1.57)	537,397 (0.08$- 1.57$)	537,397 (0.08$- 1.57$)
finish()	216,921 (0.03-0.63)	151,697 (0.02-0.44)	151,674 (0.02$- 0.44$)	151,687 (0.02$- 0.44$)
cancel()	44,462 (0.01-0.13)	116,385 (0.02-0.34)	174,950 (0.03$- 0.51$)	643,982 (0.09$- 1.88$)

- The *deploy()* and *finish()* functions are cheaper in the Multiparty protocol than in the Two-party protocol because we have optimized the code presented in Chapter 8 for the two-party protocol, making the multiparty protocol more efficient in any case.

- With respect to the Multiparty protocol, the *deploy()* and *finish()* functions cost the same because the former do not depend on the number of receivers, and the latter can only finish the protocol for one receiver. On the other hand, in the *cancel()* function we can cancel the delivery for a variable number of receivers.

The cost of this analysis is computed in gas. To determine the exact price of these transactions, we need both the gas price, set in Ethers (ETHs), and the Ether to US-Dollars exchange rate, which are variable. Therefore, the US-Dollar prices listed in Table 9.3 and Table 9.4 are provided as reference, based on the exchange rate on February 21, 2019, considering gas prices of 1 Gwei and 20 Gwei. The main difference, besides the total transaction cost, is the transaction confirmation time. A transaction with 1 Gwei can take nearly 100 minutes (depending on network traffic), while with 20 Gwei it

can be reduced to 30 seconds[3]. For more accurate comparisons with other results, it is recommended to use gas units.

9.6 Conclusions

Existing legislation lays down the rules for electronic identification and trust services for electronic transactions. Then technical proposals must achieve the legal requirements to be qualified. The features of Qualified Electronic Registered Delivery, one of the trust services included in the regulation, are similar to those offered by fair exchange protocols: Non-Repudiation of Origin and Non-Repudiation of Reception together with the integrity of the data. For this reason, it is possible to design a fair exchange protocol for registered e-delivery or certified notification. However, these kinds of services usually rely heavily on the use of Trusted Third Partys and are costly and inefficient and the behavior of the TTP has to be verified.

We have proved that blockchain-based technologies can be very useful in the design of a qualified certified notifications service, solving the problems related to the use of TTPs.

The main conclusions of the new solution are:

- It is possible to create both a solution that registers the delivery and also a proposal that protects the confidentiality of the delivered data. The choice affects strongly the design of the protocol.

- A multiparty protocol is much more efficient than a two-party protocol. The performance analysis shows that even for only two receivers the multiparty protocol requires less gas than the use of the two-party protocol.

- Using smart contracts it is possible to achieve the ideal properties without the intervention of a TTP or with minimal involvement.

- How, when and by whom a smart contract is deployed depends on the protocol. Another difference is who pays for the service.

- The use of a factory to deploy the smart contract is well suited for the non-confidential protocol but is useless in the confidential protocol. Instead, this protocol makes use of message identifiers.

- The results of the analysis of properties prove that the protocols achieve the ideal properties of the service: effectiveness, fairness, timeliness, non-repudiation, verifiability and confidentiality, as it is summarized in Table 9.5.

- The results of the performance analysis are useful to compare the protocols in terms of efficiency. The two proposed protocols differ in the amount of gas required to execute the functions of the smart contracts. Moreover, we have to take into account that some functions are not mandatory to finish the exchange.

[3] https://ethgasstation.info/

Table 9.5: Comparison of Properties for the Multiparty Certified Notifications Protocols

Property	**Non-Confidential Multiparty Protocol**	**Confidential Multiparty Protocol**
Effectiveness	YES	YES
Fairness	STRONG	WEAK
Timeliness	WEAK	STRONG
Non-repudiation	YES	YES
Verifiability	YES	YES
Confidentiality	NO	YES
Evidence Transferibility	YES	NO
TTP	VOID	OPTIMISTIC STATELESS

- The price to execute the smart contract and the time required for the validation of the transactions are related to the gas price. This way, for a low gas price, the certified notification can be executed at a cost of only a few cents of a dollar. However, the delay will be greater than an hour. Incrementing the gas price we have obtained validation times of a few seconds, while the execution costs are increased to more than a dollar. In conclusion, the delay and the cost of the certified notification can be controlled by adjusting the gas price.
- Blockchain-based technologies have been proven useful in the design of fair exchange protocols, so they may be used for the design of protocols for different services.
- The role of Trusted Third Partys can be affected by the incorporation of blockchain-based technologies in the design of protocols.

CHAPTER 10

CONFIDENTIAL MULTIPARTY CERTIFIED NOTIFICATIONS PROTOCOL WITHOUT TTP

In Chapter 8 and Chapter 9, we have discussed blockchain-based methods for certified notification services, where users receive the sender's proof of message dispatch and electronic verification of delivery attempts, that is, Non-Repudiation of Origin (NRO) evidence against Non-Repudiation of Reception (NRR) evidence. For a detailed discussion on the properties of certified notifications, refer to section 5.1.

While the earlier chapters covered two-party and multiparty protocols requiring a Trusted Third Party for confidential notifications, this chapter introduces a new protocol that ensures confidential notification without the involvement of Trusted Third Partys. The reliance on TTPs can pose technical problems like bottlenecks and delays, increase operational costs, and introduce security risks if vulnerabilities are present. Furthermore, TTPs need to be universally reliable and applicable across various legal frameworks, which is not always feasible.

The protocol proposed here combines the best attributes of the previous protocols. It allows for the certified notification of confidential data without a TTP, avoiding the technical, financial, and security disadvantages inherent in TTPs systems.

This chapter will detail a novel approach that enables both confidential data delivery and the provision of Non-Repudiation of Reception and Non-Repudiation of Origin proofs without compromising the benefits previously achieved. It eliminates the need for Trusted Third Partys entirely, thus enhancing the protocol's efficiency and security.

10.1 Contribution

In Chapter 9, two protocols have been presented. Each one of them satisfied interesting requirements. While the first proposal has allowed the complete execution of the delivery or notification operation without the implication of any Trusted Third Party for non-confidential certified notifications, the second proposal has allowed confidential certified notifications thanks to the possible use of a Trusted Third Party. Thus, to

send a notification, users first had to select if they would like to send a secret or a public notification. In both cases, the sender and receivers run a three-step exchange to transmit the content of a notification together with non-repudiation evidence.

We can summarize the protocols as follows:

- First step: the sender presents a new notification and submits it in a hidden mode to all recipients. Some parameters are incorporated in this step to guarantee the fairness property of the exchange, related to the content of the notification and the non-repudiation evidence.

- Second step: Every recipient is able to decide whether he wants to receive or not the notification. The ones who had accepted the message have to issue a Non-Repudiation of Reception evidence.

- Third step: the sender is able to finish the protocol providing a way to get the plain content of the notification and the Non-Repudiation of Origin evidence for all recipients who had accepted the notification.

The execution of the three-step protocol in the confidential scheme was *off-chain*, since the sender and recipients exchanged messages directly. Only, in case of problems, to ensure the security properties, the sender was able to cancel the exchange using a smart contract deployed specifically for this issue. Also, if receivers cannot successfully conclude the exchange, they are able to contact the TTP which will check the status of the certified notification with the assistance of the smart contract and, depending on the result, it will issue alternative evidence to guarantee fairness. Therefore, an additional conflict resolution protocol was specified between recipients and TTP to solve the exception cases. The actions of the TTP were recorded on the blockchain using the smart contract. The interaction of the confidential protocol was depicted in Figure 9.2 (section 9.2).

Whereas, in the non-confidential certified notifications protocol, all users execute the three-step procedure *on-chain*, they call the smart contract functions to perform and provide evidence of each action.

This new proposal achieves the fulfillment of the best part of each one of the previous protocols in a single protocol. This way, the protocol allows the delivery of confidential data without the need for Trusted Third Partys.

In order to achieve confidentiality in an exchange that is publicly managed by a smart contract and, as a method for the improvement of the previous proposal, we have determined that the protocol has the following requirements:

- The delivered data must be encrypted until the acceptance by the receiver when the Non-Repudiation of Reception proof is provided by the receiver.

- The smart contract cannot access the key required to decrypt the delivered data. The key cannot be included in clear in a transaction.

- The smart contract must ensure that the receivers will be able to decrypt the data after acceptance of the delivery.

- Since the protocol is multiparty, the smart contract must ensure that all the receivers decrypt the same data.

These requirements must be achieved together with the requirements to obtain fairness and the other desired properties.

Next, we present the protocol, the description of the cryptographic operations used in it and the implemented application with the smart contracts. The new protocol will be analyzed both in terms of performance and security.

10.2 Protocol design

In this proposal, confidentiality is required. The solution provides fairness to the exchange of elements: message and non-repudiation proofs even when the smart contract does not know the content of the delivered data and the plain message is not registered on the blockchain. The sender, A, executes the first step of the protocol by means of the DApp in order to register the encrypted data on the blockchain. The encryption has to guarantee that only the receivers can get access to the content of the notification. Moreover, due to the nature of the multiparty notification service, this step has to be designed in a way that the smart contract executes a verification to ensure that the message that each receiver can decrypt has the same content. To execute this step, all users have to generate a pair of keys, which will be called *Notification Keys*. The receivers, members of B, have to accept the notification by means of a transaction. The transaction is stored in the blockchain. Finally, A will execute a new transaction finishing the exchange in the third step of the protocol.

The cryptographic algorithms used in the design of the protocol are:

- ElGamal Encryption. The encryption and decryption processes are performed *off-chain*.

- Schnorr Zero-Knowledge Proof (ZKP). The verification of the ZKP is performed *on-chain* by the smart contract. The description of the solidity implementation of the ZKP is included in this chapter. For a more detailed description of the Schnorr ZKP, see subsection 4.7.5.

The ZKP proof is introduced in the protocol to check whether all receivers are able to decrypt the same notification content or not and, at the same time, the scheme preserves the confidentiality of the delivered data. In this way, the sender commits to sending a key to all the receivers during the creation of a new notification. This key will allow each receiver, who has accepted the message, to open its content. Thus, the key, instead of being published, is encrypted with every *secret shared notification key* that it is only known by each receiver and the sender. The ZKP introduced in the protocol allows the smart contract to check that the key to open the message is the same for all the receivers without knowing the bit string of the key. Therefore, the protocol preserves the confidentiality between sender and receiver and, at the same time, it can publicly verify that each receiver has access to the same content.

Cryptographic Background and Notation

Proper parameters should be published prior to the use of the protocol. Also, users have to be able to check the correctness of these domain parameters. The appropriate operational conditions, before starting with the *Creation* phase of the protocol as it is described in the next section, are the following:

- Two large primes have to be published: p and q with $q|p-1$ (q is a primer large factor of $(p-1)$)
- The operations will be made in $GF(p)$ and in Gq. Where Gq denotes the subgroup of the multiplicative group of $GF(p)$, of prime order q
- Let g be a generator for the subgroup Gq, such that $1 < g < p$
- Random numbers r and s are uniformly chosen between 0 and $(p-1)$
- Private keys x_i are randomly (or pseudorandomly) generated from $[0, q-1]$
- Public keys are created as: $y_i = g^{x_i} mod p$
- Fragments of the message $M[j]$ to be encrypted are in the range $0 < M[j] < p$

The notation used in the description of the protocol is included in Table 10.1.

Phases of the Protocol

The exchange protocol that ensures the fairness of the exchange and fulfills the confidentiality requirements listed in section 10.1 includes three phases. These phases are called *Creation*, *Acceptation* and *Finalization*. A fourth phase, *Cancellation*, is optional. There is also a *Verification* process to check the status of the exchange. The main flow of the protocol is depicted in Figure 10.1. In this Figure, the three actors are represented at the top (Sender, Blockchain and Receivers) and, as it is illustrated in the outline, all phases are executed *on-chain*. Before the deadline $term_1$, there are depicted phase 1 (*Creation*), which is executed by the sender, and phase 2 (*Accept*) executed by all the receivers who agree to get the notification. In the Figure, between the two red lines, which represent the deadlines $term_1$ and $term_2$, there is the *Finish* phase that is performed by the sender for each particular receiver and, at the end of this phase, the receiver is able to decrypt the notification content. If the decryption has not been successful, there is an optional phase (*Cancellation*) that can be executed by any receiver to conclude the notification in a fair way. The phases are described in this section.

1. **Creation** Subprotocol 1. A encrypts the message M in cipher text C using a variant of ElGamal encryption and the Discrete Logarithm Integrated Encryption Scheme (DLIES)[1]. To do this, the sender A uses a secret encryption element r to generate the final key to encrypt the message M using an XOR cipher operation. If it is necessary, the message can be fragmented in $M[j]$ and, thus, the result of the encryption will be the $C_2[j]$ fragments. Then, C_1 represents the bit commitment with secret encryption element r that will be used by the Smart Contract

[1] Encryption method standardized in ANSI X9.63, IEEE 1363a, and ISO/IEC 18033-2

Table 10.1: Notation of the confidential multiparty certified notifications protocol without TTP

Notation	
A	Sender.
B	Set of receivers. B_i is used for a single receiver.
B'	Set of receivers that have accepted the delivery
M	Message, content of the notification.
$M[j]$	Fragment of the message.
X, Y	Concatenation of messages X and Y.
$U \blacktriangleright e.f$	Execution of function f of e by user U.
$term_1$	Timeout for B_i to accept the notification.
$term_2$	Timeout for A to finish the exchange.
D	Deposit sent to the Smart Contract (ethers).
x_A	A's notification private key.
y_A	A's notification public key.
x_{Bi}	B_i's notification secret shared key.
y_{Bi}	B_i's notification public key.
g, p, q, g	System parameters.
r	Encryption secret nonce used by A.
s	Encryption nonce used by B.
$C_1 = g^r_{modp}$	First part of ElGamal encryption of the data to deliver.
$C_2[j] = M[j]$ XOR key	Second part of ElGamal encryption of the data to deliver.
$key =$ $random.seed(hash(r))$	Encryption key
$h()$	Hash Function
$challenge$	Challenge sent by B to A
w	Response to the challenge.
$Z_{i1}=g^{s_i}{}_{modp}$	First part of the encryption of the receiver notification secret key
$Z_{i2}=x_{Bi} * y_A^{s_i}{}_{modp}$	Second part of the encryption of the receiver notification secret key

during the $Finish$ phase to verify that all receivers have access to the same r and, therefore they can decrypt the same content of the notification.

To generate a new notification, sender A has his own pair of notification keys created specifically for this exchange, (x_A, y_A). Next, A creates a new instance of a smart contract to manage the new notification delivery by invoking the factory constructor function provided by the service provider, including the following parameters: the encrypted message $\{C_1, C_2\}$, the public notification key y_A, the addresses of the proposed receivers (the set B) and the deadlines $term_1$ and $term_2$. The first one ($term_1$) is the deadline for each receiver to accept the delivery. The second deadline ($term_2$) specifies the valid period for the sender to finish the exchange. Also $term_2$ designates the moment since when the receivers will be able to have evidence of origin and the plain message or, if the protocol run has not finished successfully, they can obtain evidence of the cancellation of the process that has not been properly completed by the sender. At this point,

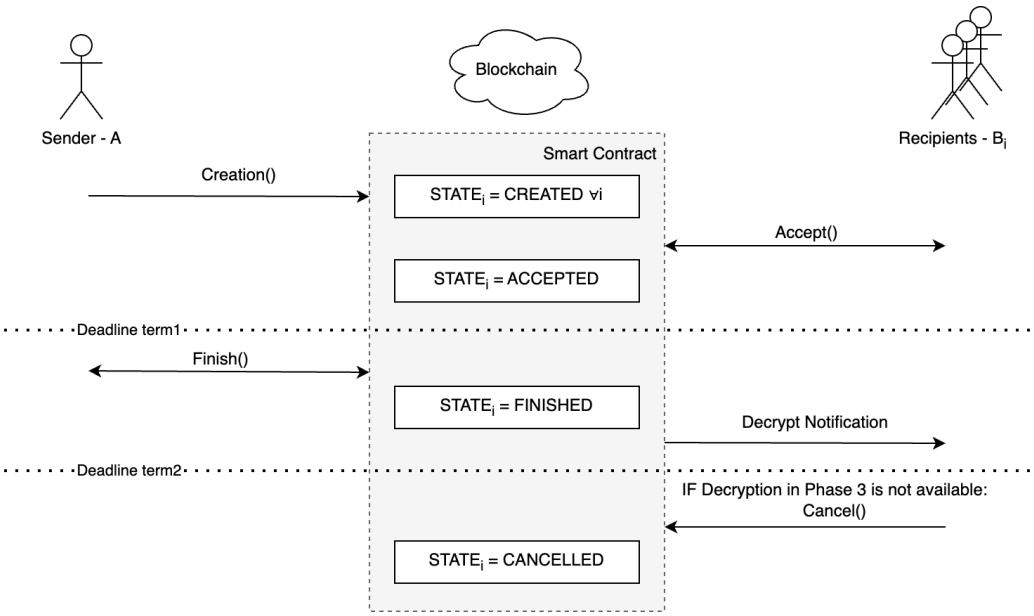


Figure 10.1: Description of the confidential multiparty certified notifications protocol without TTP

we want to highlight the purpose of the parameter C_1: the parameter represents the bit commitment of sender A with the encryption secret key r, in such a way that by making public this parameter (C_1) the smart contract can publicly verify that the sender is sending the same key to all recipient and, the key is the one that A is being committed to send once C_1 has been made public. We will see how, in our protocol, A sends r in an encrypted envelope created with a shared secret key between each receiver and the sender. These shared keys are created by each recipient and they are confidentially sent to the sender in the $Accept$ phase. Then, at the $Finish$ phase, the smart contract will check that C_1 contains the right key r using the Schnorr ZKP primitive without the need to know r.

A payment for the service or a deposit D can be included in this stage. Even if fairness is ensured in any case, in order to avoid dishonest behavior and fraud attempts, the protocol includes a penalty mechanism to avoid this behavior from bothering other users, in terms of delay or differences in the distribution of execution costs. For this reason, we have included in this phase of the protocol a deposit with the aim of encouraging the sender to conclude the exchange in the expected way, that is, following the phases of the protocol. As it will be explained in the *Finish* phase the deposit D will be returned to the sender if he finishes the exchange according to the protocol.

2. **Accept** Subprotocol 2. In this multiparty scenario, each receiver can decide individually whether to accept the delivery or not, by executing the corresponding function of the smart contract before the deadline $term_1$. If a receiver accepts the delivery, he executes a function of the smart contract expressing his will. If the receiver B_i does not accept before $term_1$, a rejection is assumed ($State_i = Rejected$), otherwise, the delivery has been accepted by the receiver

Subprotocol 1: Step 1. Creation

1. A:
   Generation of $M, r, y_A = g^{x_A}_{modp}$
   $key = random.seed(hash(r))$
   Encryption of M.
   If required, fragmentation of M in blocks: $M[j]$
   $C_1 = g^r_{modp}$
   FOR $j = 1$ **TO** $M.length$
   $C_2[j] = M[j]$ XOR key
   $key = random.seed(hash(key))$
2. $A \blacktriangleright SM.creation(A, B, C_1, C_2, term_1, term_2,$
$y_A, g, p, q, D)$
3. SM:
   $State_i = Created, \forall i$

B_i ($State_i = Accepted$). Since A has to allow access to the delivered contents to those members of B that have accepted the delivery while keeping it confidential for the rest of the world, the plain message cannot be included in a transaction nor stored in the blockchain. Thus, the sender A must generate the necessary elements to confidentially deliver the key to decipher the message, (that is, sending the key that has been committed to send at the *Creation phase*, C_1). The smart contract must ensure that all the receivers that have accepted the delivery can access the same message. In order to achieve this goal, each receiver B_i generates its own pair of shared notification keys x_{Bi}, y_{Bi}. This pair of keys is called $shared$ because they will be shared between a particular recipient and the sender. While x_{Bi} is the private key because it is only known by the recipient and the sender, y_{Bi} is the public key because everybody can know it. This public shared notification key will be used in the next phase ($Finish$) by the smart contract to check whether all receivers are able to decrypt the same message and, thus, they can read the same notification content. Accordingly, B_i generates parameters $\{Z_{i_1}, Z_{i_2}\}$ that will allow sender A to get the shared key x_{Bi} as we will see at the $Finish$ phase. The calling to the function $accept()$ also includes the parameters y_{Bi} and $challenge_i$. The latter is a uniformly random chosen challenge that the sender should appropriately respond in conjunction with the secret encryption element r. The response will be used by the smart contract to verify that every recipient receives the same key r at the $Finish$ phase. Note that, in spite of being able to verify the correctness of the secret r, the smart contract will not be able to know the secret r thanks to the use of the ZKP primitive.

Since each receiver is free to accept the notification or not, he can't be punished if he does not accept the notification. For this reason, no penalty mechanism has been included in this phase of the protocol.

3. **Finish** Subprotocol 3. Finally, before the deadline $term_2$, A can finish the delivery process for those B_i who have accepted the notification, executing the *finish()* function of the smart contract.

Subprotocol 2: Step 2. Accept

1. B_i :
   generation of the parameters: $y_{B_i} = g^{x_{B_i}}_{mod p}$, s_i
   Z_{i_1}=$g^{s_i}_{mod p}$, Z_{i_2}=$x_{B_i} * y^{s_i}_{A\, mod p}$
2. B_i ► $SM.accept(Z_{i_1}, Z_{i_2}, y_{B_i}, challenge_i)$
3. SM:
   IF$(now < term_1)$ AND $(Id == B_i)$ AND $(State_i == Created)$
   $State_i = Accepted$
   Add B_i to B'

The sender A generates for each receiver in B', that is, receivers that have accepted the delivery, a response to the challenge in the form of a ZKP using the secret element r used in the encryption of the message in the first step of the protocol, the challenge provided in phase 2 and B_i's secret shared notification key, x_{Bi}. Note that, although the secret shared notification key, x_{Bi}, was created by B_i, B_i has sent the secret shared key to A, encrypted with A's public key using ElGamal cryptosystem, resulting in $\{Z_{i_1}, Z_{i_2}\}$. This way, A can obtain the secret shared notification key for each B_i using its private key: $\left({Z_{i_1}}^{x_A}\right)^{-1} * Z_{i_2} = x_{Bi}$.

The Smart Contract will store the parameters. With this response, the Smart Contract can verify, by means of the stored data, that each receiver B_i in B' will be able to know the secret element r in order to decipher the message. However, the Smart Contract will not know this element and therefore the message will remain confidential. Thus, the ZKP allows the Smart Contract to verify the commitment with the secret key r that did A in the $Creation$ phase of the certified notification, in such a way that A can prove to the Smart Contract that he is sending the proper secret element r to each receiver without disclosing its value, because his response w_i to the challenge sent by any receiver $challenge_i$ is coherent with the committed value of the secret key, publicly expressed in C_1. Therefore, if $g^{w_i} == g^r * y_{Bi}^{challenge_i}{}_{mod p}$ holds, then the Smart Contract can be sure that the receiver is able to get access to the right key and the state of the notification for this receiver will be $Finished$.

In the last step of this phase, receivers can isolate r from w due to knowledge of x_{Bi} and $challenge$ and, then, they will be able to read the content of the message.

At this point, if the sender has executed *Finish* in the right period of time, and if the verification performed by the smart contract stands, since the sender A has followed the protocol, the smart contract returns the amount D, deposited in the *creation* phase, to the sender A. With this procedure, the protocol wants to avoid misbehavior from A. If at *term2 Finish* has not been executed satisfactorily, then the deposit D will not be returned. As it will be explained in phase *Cancel*, in this situation, a receiver that has accepted the notification can execute *Cancel* and the exchange will remain in a fair situation. However, this has to be executed

Subprotocol 3: Step 3. Finish

1. A :
   $x_{Bi} = \left(Z_{i_1}{}^{x_A}\right)^{-1} * Z_{i_2\, mod p}$
   generation of w_i=$r + challenge_i * x_{Bi\, mod q}$
2. $A \blacktriangleright$ SM.finish(w_i)
3. SM:
   IF ($Id == A$) AND (($term_1 < now < term_2$) OR (($B' == B$) AND ($now < term2$))
      FOR ($\forall B_i \in B'$)
         IF ($g^{w_i} == g^r * y_{Bi}^{challenge_i}{}_{\, mod p}$)
            $State_i = Finished$
      FOR ($\forall B_i \notin B'$)
         $State_i = Rejected$
      Deposit D is refunded to A
4. B_i :
   $r = w_i - challenge_i{}^* x_{Bi\ mod q}$
   $key = random.seed(hash(r))$
   FOR $j = 1$ **TO** n
      $M[j] = C_2[j]$ XOR key
      $key = random.seed(hash(key))$

after *term2* and it is the receiver who has to execute the function in the smart contract, causing delay and a different distribution of the execution costs. It has to be said that if no receiver accepts the notification then the deposit D is also returned to the sender.

Finally, after $term2$, each receiver in B' (receivers that have accepted the delivery) can access the message through the *Web3* or similar interface or, in the event that the issuer has not successfully completed the protocol, the receivers will have access to the corresponding cancellation evidence.

4. **Cancellation** Subprotocol 4. Cancellation of acceptance. This step is optional and it will be executed by any receiver B_i, if the sender A does not finish the exchange providing the decryption key when the receiver has accepted the contract.

   If a receiver tries to cheat by executing *Cancel* when the state is not *Accepted* (then it is *finished*), the protocol will not change the state, maintaining fairness. However, the dishonest behavior of this receiver can be punished by including him in a blacklist of dishonest users.

5. **Verification** The verification process can be carried out by both the sender A and any receiver B_i, as well as any third party involved. In this verification process, the status of each receiver can be checked, with the *getState* function, as well as any variables involved in the exchange, as they are public in the smart contract, except the message M, which for reasons of confidentiality, is encrypted.

Subprotocol 4: Cancellation of Acceptance

1. B_i ► SM.cancel()
2. SM:
   IF($now >= term2$, $Id == B_i$ AND $State_i == Accepted$)
   $State_i = Canceled$

In addition, as the verification of the ZKP is performed on-chain, in the *Finish* function, we can ensure that the state of any receiver B_i is checked by the contract itself.

10.3 Implementation

The protocol presented in section 10.2 has been implemented to prove its viability and also to check the execution costs. For this implementation, we have used the Ethereum blockchain. This will offer us more functionalities than conventional blockchains such as Bitcoin, using programs called *Smart Contracts*, which allow us to program on a distributed *Turing complete* machine, developed in *Solidity* language, and to store system state changes. Ethereum will use *Ether (ETH)*, its cryptocurrency, to meter and limit the costs of resources used to execute the code. This section includes the development procedure and the description of the smart contracts.

In this protocol, a *Smart Contract* is used to manage the distribution of a confidential message from a sender A to several recipients (all recipients are the members of a set called B) and to exchange Non-Repudiation of Origin evidence against Non-Repudiation of Reception evidence.

To do this, we need some data structures to keep the group of recipients, thus an array is declared to keep their addresses. Also, a mapping structure has been declared to keep track of the state of each exchange for every recipient, and the variables *z1*, *z2*, *yb*, *c* and *w* (that correspond to the elements Z_{i1}, Z_{i2}, y_{Bi}, $challenge_i$ and w_i according to the notation of the protocol) of the implementation of the ZKP, for each recipient. This mapping allows us to maintain the cast and the time for searching the information from a receiver as constant values. On the other hand, the array lets us to iterate through all the addresses of the receivers.

In addition to that, two variables (*term1* and *term2*) are also declared in the smart contract to store the correspondent deadlines and, also, an additional variable (*sender*) to save the sender's address. We have defined the variable called *start* to set the current time when a delivery is created, and the deadlines *term1* and *term2* are set from this value. A new variable *acceptedReceivers* is used to count the amount of recipients that have accepted the delivery. This way, the sender can finish the delivery of the data and he does not need to wait until *term1*. Thanks to the *acceptedReceivers* variable, the smart contract does not have to verify the state of the exchange from all the receivers, thus we are avoiding an expensive operation in terms of gas consumption thanks to this variable.

We also need to store all the general variables of the Zero-Knowledge Proof (ZKP): *c1*, *c2*, *ya*, *g*, *q* and *p* (C_1, C_2, y_A, g, q and p). All data structures are written down in

```
enum State {notexists, created, canceled, accepted, finished, rejected }

struct ReceiverState{
  bytes z1;
  bytes z2;
  bytes yb;
  bytes c;
  bytes w;
  State state;
}

address public sender;
address[] public receivers;
mapping (address => ReceiverState) public receiversState;
uint acceptedReceivers;

bytes public c1;
bytes public c2;
bytes public ya;
bytes public g;
bytes public p;
bytes public q;

uint public term1;
uint public term2;
uint public start;
```

Listing 16: Data structures

```
function accept(bytes _z1, bytes _z2, bytes _yb, bytes _c) public {
  require(now < start+term1, "The timeout term1 has been reached");
  require(
    receiversState[msg.sender].state==State.created,
    "Only receivers with 'created' state can accept");

  acceptedReceivers = acceptedReceivers+1;
  receiversState[msg.sender].z1 = _z1;
  receiversState[msg.sender].z2 = _z2;
  receiversState[msg.sender].yb = _yb;
  receiversState[msg.sender].c = _c;
  receiversState[msg.sender].state=State.accepted;
}
```

Listing 17: accept() function

Listing 16.

The following variables are initialized when a new instance of the *Smart Contract* is created: first, an array of recipients is initialized with the values sent by the sender, and next the delivery's state for each recipient is set to *created*. We will also store the timeouts and all the general variables for the ZKP and the sender makes a deposit in Ether.

The *Accept* function verifies that the user who has called the function (*msg.sender*) has the address in the recipients mapping (see Listing 17), and the correspondent

```
function finish(address _receiver, bytes _w) public {
  require((now >= start+term1) || (acceptedReceivers>=receivers.length),
    "The timeout term1 has not been reached and not all receivers have accepted
        the delivery");
  require (msg.sender==sender, "Only sender of the delivery can finish");

  bytes memory check_1=bignumber_modexp(g, _w, p);
  bytes memory check_2=bignumber_modmul( c1,
      bignumber_modexp(receiversState[_receiver].yb,
      receiversState[_receiver].c, p), p);

  require (bignumber_equals(check_1, check_2),
    "(g^w mod p) and (((g^r mod p)(ybc mod p)) mod p) are not equals");
  sender.transfer(this.balance);
  for (uint i = 0; i<receivers.length; i++) {
    receiversState[receivers[i]].w = _w;
    if (receiversState[receivers[i]].state == State.accepted) {
      receiversState[receivers[i]].state = State.finished;
    } else if (receiversState[receivers[i]].state == State.created) {
      receiversState[receivers[i]].state = State.rejected;
    }
  }
}
```

Listing 18: finish() function

delivery state is *created*. Then, the function also verifies that *term1* deadline has not been overtaken. The delivery state of the recipient will be updated to *accepted* if all verifications are fulfilled, and all variables for the ZKP which depends on each receiver are stored for that particular receiver.

The *Finish* function is depicted in Listing 18. This function first verifies that the current time is greater than *term1* or whether all recipients have accepted the exchange or not. So as to carry on with the function, it is also necessary to check that the user who has invoked the function is the sender of the notification. Finally, the function also checks that the w variable fulfills the equality $g^w mod\ p = (c1 * ({y_b}^c\ mod\ p))mod\ p$, using the solutions explained in section 10.3. Then, if all of these verifications are met, the delivery states of all the recipients that have been *accepted* are updated to *finished*. Also, for the recipients whose state is *created*, the state is changed to *rejected*, and the sender will be refunded.

Lastly, we have depicted the *Cancel* function in Listing 19. This function first

```
function cancel() public {
  require(now >= start+term2,
      "The timeout term2 has not been reached");
  require(receiversState[msg.sender].state==State.accepted,
      "Only receivers with 'accepted' state can cancel");

  receiversState[msg.sender].state = State.canceled;
}
```

Listing 19: cancel() function

verifies that the current time is greater than the deadline *term2* and that the caller of the function is one of the recipients whose delivery state is *accepted*. If these conditions are met, then the smart contract updates the receiver's delivery state to *canceled*.

Factory Contract

We have also used the well-known factory method programming pattern in order to generate new instances of a smart contract to manage each newly certified notification. We have created the *Factory Contract* that generates and deploys new contracts for this purpose, the use of this pattern will achieve the following advantages:

- The code used to generate and manage a newly certified notification will be reusable.
- The factory also works as a storage of the addresses of the child contracts, the certified notifications.
- We can easily access all the notifications that we have made using this service thanks to this new address storage inside the factory contract.
- The provider of the central service only has to pay for the deployment of the factory contract.
- Each user who wants to send a new notification has to pay for the creation of a new instance. We have analyzed the deployment costs in section 10.5. According to this pattern, the factory contract is in charge of the deployment of new instances to manage a new notification and it will keep the address of this newly deployed contract in an inner data structure.

ZKP in Solidity

Implementing the ZKP in solidity is challenging due to the use of big numbers. An inefficient implementation would result in high execution costs in terms of gas.

The most important problem found to implement the Zero-Knowledge Proof in Solidity is that we need to operate with numbers of minimum 256 bits, and there is an operation that checks an equality, $g^w \, mod \, p = (c1 * ({y_b}^c \, mod \, p)) mod \, p$, that needs more bits to operate.

The solution found consists of using the *bytes* data type, which can hold a sequence of bytes of any size, instead of using the *uint* data type, which can hold unsigned integers of a maximum of 256 bits. If we use this data type in the parameters of the functions and to store values, we can use these Smart Contracts with numbers of any size, like 256, 512, 1024 or 2048 bits.

Then, we have also used the Big Number Library for Solidity [2] to operate with these numbers. First of all, we convert the *bytes* values to BigNumber values, and then we perform the *equals()*, *modmul()* and *modexp()* functions using this type, employing the following functions from the Big Number library:

[2] (https://github.com/zcoinofficial/solidity-BigNumber/)

1. `bignumber_equals(bytes _a, bytes _b)`: Compares two BigNumber instances for equality. It returns `true` if the two numbers are equal and `false` otherwise.

2. `bignumber_modmul(bytes _a, bytes _b, bytes _m)`: Performs modular multiplication of two BigNumber instances _a and _b with respect to modulus _m. It returns the result as a `bytes` value.

3. `bignumber_modexp(bytes _b, bytes _e, bytes _m)`: Performs modular exponentiation of a BigNumber instance _b raised to the exponent _e with respect to modulus _m. It returns the result as a `bytes` value.

10.4 Security properties analysis

Now, this section presents a survey and a security discussion of the desired properties for certified notifications schemes that were enumerated in section 5.1.

Regarding the security properties defined in section 5.1, we have removed the *efficiency* property because it will be evaluated separately, in section 10.5, where the results and a set of experiments to evaluate the performance of the protocol are presented. We also have grouped together the properties of *fairness* and *evidence transferability* for discussion purposes.

1. **Effectiveness.** The system for certified notifications presented in this chapter is effective. So, all parties will receive the expected items in case of behave according to protocol.
   To create a new notification, the sender generates a new instance of the smart contract to execute the functions following the specifications of the protocol. If all the parties execute all the functions correctly (*Accept()* and *Finish()* functions), they will have the items that they expect to receive. This can easily be deduced from the solidity code depicted in section 10.3. At the end of the exchange, the receivers that have followed the protocol will have the key to decrypt the delivered data and the Non-Repudiation of Origin proof whereas the sender will have the Non-Repudiation of Reception proof and the state of this delivery will be *Finished* for every recipient.

2. **Fairness** and **Transferability of evidence.** The proposed protocol for certified notifications is fair. At the end of a protocol execution, either each party has received the proper element or neither party has received any useful data about the other's element, providing *strong fairness*[87]. Moreover, the evidence generated by the protocol can be transferred to an external party in order to prove the outcome and the effects of the exchange without further verifications.
   On the one hand, according to the protocol, the sender is not going to receive the Non-Repudiation of Reception evidence generated by the smart contract except if he executes a transaction in order to allow the receivers to decrypt the message and, at the same time, allows the smart contract to check the validity of the provided key (when *state* = *Finished*). On the other hand, any receiver B_i is able to get access to the delivered data only if he runs a transaction in order to agree to receive the notification (case *state* = *accepted*).

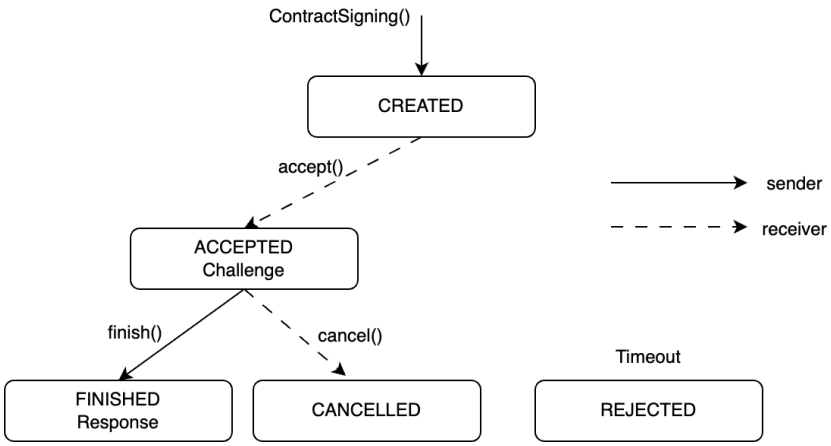


Figure 10.2: States of the confidential multiparty certified notifications protocol without TTP

If the parties do not follow the specifications of the protocol, that is, if they do not execute the functions *Accept()* and/or *Finish()*, the smart contract guarantees a result that is fair for every user without the need of any intervention of a TTP.

We have analyzed all the cases where the protocol can lead the exchange to prove the strong fairness of the protocol. We have used a state transition diagram. For each receiver, this diagram is formed by three final states: *Finished*, *Canceled* and *Rejected* and two intermediate states: *Created* and *Accepted*. These states cannot be final states because the protocol will eventually change the state in any case. The states and the transactions are represented in Figure 10.2.

Finished implies that the exchange has been completed following all the stages of the protocol, while *Canceled* and *Rejected* represent exchanges that have not been completed, for different reasons. Now, we will show how the protocol leads the exchange to a fair situation in all cases:

- *creation() not executed*. If the first step of the protocol is not executed satisfactorily, the notification is not created, any element of the fair exchange has been provided and the parties are in a fair situation.
- *accept() not executed satisfactorily*. If, after the creation of the notification in the first step of the protocol, a receiver does not execute *accept*, before the deadline *term1*, then the smart contract will set the state for this receiver to *Rejected*. This way, after the first timeout, those receivers that have not called the *accept()* function are not included in the set B' and the sender will not finish the exchange with these receivers, so the delivery will not be completed for them and they will not receive the decryption key. These receivers do not have access to the delivered data and to the non-repudiation of origin element. On the other hand, the sender does not have a Non-Repudiation of Reception proof generated by these receivers. So, the situation is fair for all the parties. The final state for this exchange will be *Rejected*.

- *finish() executed successfully.* The sender can execute *finish()* only for those receivers that have executed *accept()*, that is, the receivers in B', with state equal to *accepted*. The sender can, consequently have the Non-Repudiation of Reception proof. When the sender executes *finish()* then the smart contract checks the provided elements and verifies that the receiver will have access to the delivered data and also that he obtains the Non-Repudiation of Origin proof (state==*finished*). This way, after successful execution of *finish()*, the protocol leads the exchange to a fair state, where the parties have received all the desired elements.
- *finish() not executed successfully. Execution of cancel().* If after the execution of *accept()*, a receiver does not receive the decryption key (that is, the sender has not executed *finish()* or the smart contract does not verify the provided elements), then, after the expiration of the deadline, the receiver can call the *cancel()* function to conclude the exchange with fairness. This way the smart contract will set the state to *Canceled*, meaning that the exchange has not been performed successfully, that is, the receiver hasn't had access to the delivered data and the protocol has not generated non-repudiation proofs.

According to the previous evaluation, the fairness provided by the protocol is *strong fairness*. Although the smart contract can create finalization and alternative cancellation proofs, the protocol does not allow any circumstance where a user could get contradictory evidence since the state for each receiver is only updated by the smart contract. Therefore, it is not possible to do an action that could lead to an unfair situation (the contrary would be considered *weak fairness* [87]).

The proofs generated during the execution of a protocol run can be submitted as proofs or evidence to an external arbiter. We have to take into account that users cannot get contradictory proofs and valid evidence are only created by the execution sequence of the smart contract. The evidence can be evaluated by an external arbiter who can determine whether the exchange has concluded successfully or not. In addition to that, its transferability is easy, because the proofs generated during the exchange are all stored on the blockchain. Therefore, since the blockchain is immutable, it is not possible to change the content of any message and, thus, the scheme provides message integrity. It is also possible to deduce the exact point in time when the delivery took place from the timestamp of the block where the transaction was included.

3. **Temporal parameters: Timeliness and Timestamping.** A successful delivery will always be completed before deadline *term2*.
If the delivery is not successful we have different deadlines in function of how the exchange has been performed. If a receiver does not accept the delivery, then the delivery will be classified as *Rejected* at *term1*. If after the acceptance of the delivery by a receiver the sender does not finish the exchange, then the exchange can be canceled at *term2*.

Moreover, the system prompts the sender to end a notification exchange before the deadline *term2*. This motivation is performed by means of a deposit that is

blocked in the smart contract. The deposit will be given back to the sender in case of conclusion before *term2*. We have to take into account that the blockchain timestamps all transactions performed on it.

4. **Non-repudiation.** The certified notifications protocol must provide a Non-Repudiation of Reception evidence and a Non-Repudiation of Origin evidence too.

    - Regarding the origin, a sender A of the notification cannot deny having executed the function to create the notification since there is a transaction on the blockchain from her address containing the addresses of the receivers and the encrypted message (see Subprotocol 1). In addition, the transaction related to the execution of the *finish()* function (see Subprotocol 3) proves that the sender has provided the key to decrypt the message to the receivers that accepted the certified notification, and is considered the Non-Repudiation of Origin element since this transaction leads the exchange to the *Finished* state.
    - Concerning the reception, each recipient B_i is not able to refuse having accepted the notification, because an accepting transaction from his address is stored on the blockchain. This transaction accepts the reception of the notification and, according to that, the smart contract changed the *state* of the notification to *Finished* after the execution of *finish()* function. The protocol ensures that each receiver has obtained the right decryption key since the smart contract validates it by using the ZKP (see Subprotocol 2 and Subprotocol 3).

5. **Trusted Third parties.** The proposed scheme does not need the intervention of an external Trusted Third Party.
   In this protocol, the parties run the functions of the smart contract by generating the appropriate transactions and, thanks to the use of the smart contract, no further dispute-resolution phase is needed. The exchange of information in the proposed system is all on-chain, as a consequence, all communications related to a notification are stored on the blockchain. We have to take into account that blockchain technology has been conceived to be immutable and publicly verifiable, thus the actions that have been carried out in a protocol run are publicly verifiable and anyone can know which address is accountable for that.
6. **Confidentiality**. A delivery will be confidential if only the receivers that accept the delivery can access the delivered data. For this reason, the data cannot be included in clear in any transaction and must not be registered in the blockchain. The data cannot be a parameter in the smart contract functions and the smart contract cannot gain access to the decryption key. Even though, the protocol must check that the data received by all the receivers is the same.
   In Subprotocol 1 the sender runs the *creation* function of the smart contract including C_1 and C_2, that represent the encrypted message. In Subprotocol 2 each receiver provides a way for the sender to send privately the key to decrypt the message, through the elements Z_1 and Z_2. The smart contract verifies in Subprotocol 3 that the right decryption key is provided and that can be derived from w_i, isolating r.

There are no more entities involved in the notification exchange, and the transactions only include encrypted messages and hidden decryption keys that are only recoverable for receivers who have accepted the exchange. Thus, the delivered data will remain confidential.

10.5 Performance analysis

Once we have finished the implementation of the proposal and we have tested the results, we have performed some experiments to determine the efficiency of the system in terms of cost, since the economical execution costs could be a concern in the development of the certified notifications service.

The performance analysis includes several tests, performed using the *Smart Contracts* explained in section 10.3. We have deployed it to the Ganache network, a personal blockchain used for Ethereum development, to isolate the performance conditions and possible problems of a real network like the main Ethereum or the Rinkeby test networks.

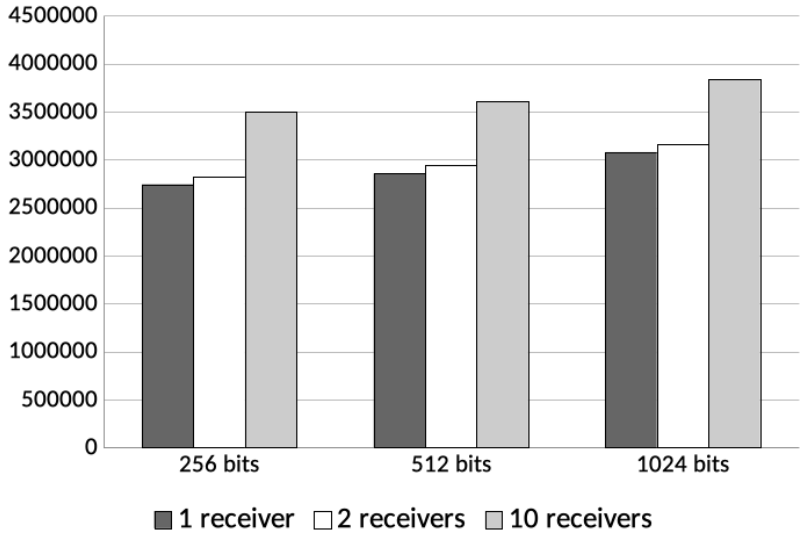


Figure 10.3: Gas cost of the createDelivery() function

Table 10.2: createDelivery() function gas cost and equivalent US-Dollars price with 1 Gwei - 20 Gwei gas price

Cost in Gas	**256 bits**	**512 bits**	**1024 bits**
1 receiver	2,743,000	2,826,871	3,077,925
	(0.35-7.09)	(0.35-7.21)	(0.38-7.78)
2 receivers	2,826,871	2,938,640	3,161,732
	(0.35-7.14)	(0.36-7.43)	(0.40-8.00)
Cost per Receiver	(0.17-3.57)	(0.18-3.71)	(0.20-4.01)
10 receivers	3,498,345	3,610,115	3,833,208
	(0.43-8.85)	(0.45-9.13)	(0.48-9.70)
Cost per Receiver	(0.04-0.87)	(0.04-0.91)	(0.04-0.96)

From the results of the tests, we have detailed the gas cost of the functions, executing the protocol with different values for the number of receivers, selecting the values of one, two and ten receivers. With these tests, we want to obtain absolute values of the economic costs to evaluate the viability of the protocol and also evaluate the performance of multiparty protocols compared to Two-party protocols and for this reason, the cost per receiver has been taken into account.

We have also tested the performance in terms of delay to provide an illustrative reference of the delays introduced by the functions, although we have not included the complete list of results in this chapter. In the Ganache network, all the functions have been executed with a delay between 55 and 837 milliseconds, closely related to the gas cost of each function. However, in a normal production Ethereum blockchain network, the delay for transaction validation varies depending on the fee the sender is willing to pay and the current network conditions. With the transition to Proof of Stake (PoS), the average delay is generally reduced compared to the Proof of Work (PoW) consensus mechanism, typically ranging between 15-30 seconds.

In Figure 10.3, Figure 10.4, and Figure 10.5, we can see the gas cost of the main functions of the protocol. The cost analysis is computed in gas, but to determine the exact price of these transactions, we also need the gas price, set in Ethers (ETHs),

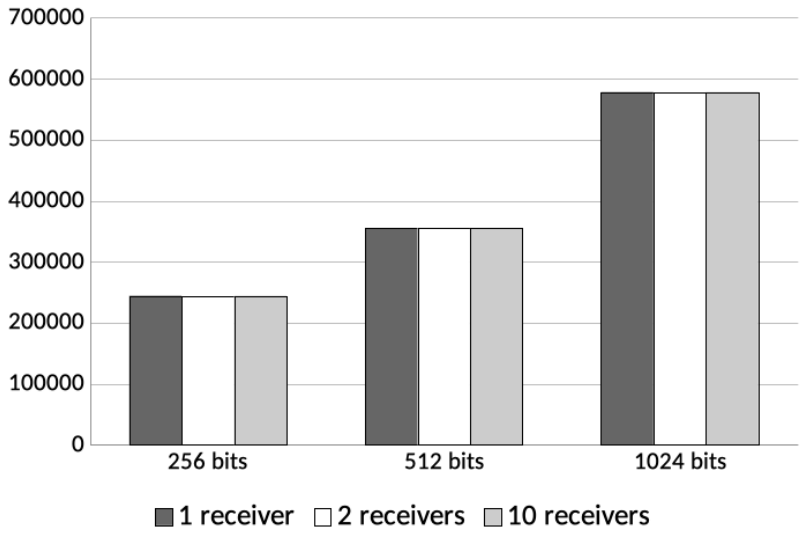


Figure 10.4: Gas cost of the accept() function

Table 10.3: accept() function gas cost and equivalent US-Dollars price with 1 Gwei - 20 Gwei gas price

Cost in Gas	**256 bits**	**512 bits**	**1024 bits**
1 receiver	244,207 (0.03-0.61)	355,403 (0.04-0.90)	577,988 (0.07-1.46)
2 receiver	244,207 (0.03-0.61)	355,403 (0.04-0.90)	577,988 (0.07-1.46)
10 receiver	244,207 (0.03-0.61)	355,403 (0.04-0.90)	577,988 (0.07-1.46)

and the Ether to US-Dollars exchange rate, both of which fluctuate. Therefore, we have added in Table 10.2, Table 10.3, and Table 10.4 the price in US-Dollars of each functionality as a reference, considering the exchange rate on January 1, 2020[3], with gas prices of 1 Gwei and 20 Gwei. The cost in gas of the *deploy()* function of the *factory* is not included, as it is independent of the number of receivers and the number of bits: 3,864,642 (between 0.49$ and 9.78$). The main difference, in addition to the total cost of the transaction, is the time it will take for the transaction to be accepted by a mining node. A transaction made on the main Ethereum network with a value of 1 Gwei can take almost 100 minutes (depending on the current network traffic), while in the case of 20 Gwei, it can be reduced to 30 seconds[4]. To accurately compare with other results, it is better to compare the gas costs directly.

From the analysis of the results included in Figure 10.3, Figure 10.4, Figure 10.5, Table 10.2, Table 10.3 and Table 10.4 we can conclude that:

- Cost of the functions:

[3]Ether Price = 130.27$
[4]https://ethgasstation.info/

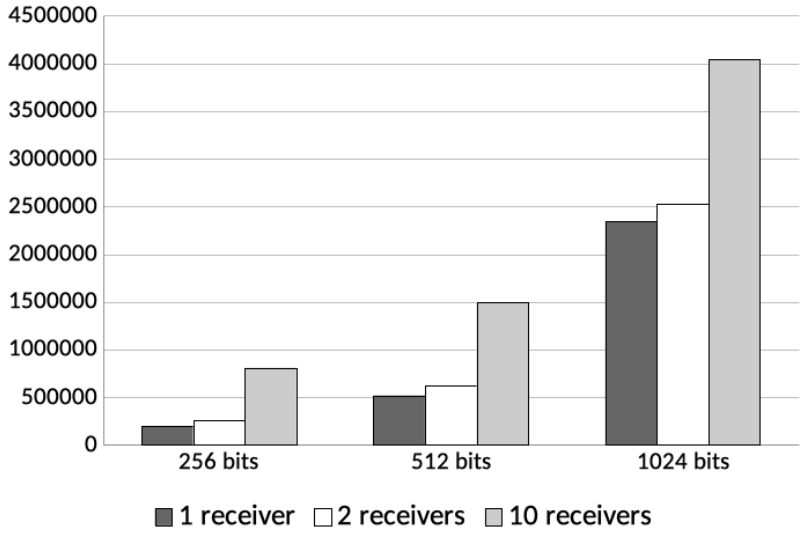


Figure 10.5: Gas cost of the finish() function

Table 10.4: finish() function gas cost and equivalent US-Dollars price with 1 Gwei - 20 Gwei gas price

Cost in Gas	**256 bits**	**512 bits**	**1024 bits**
1 receiver	192.681 (0.02-0.49)	516.563 (0.06-1.30)	2.343.311 (0.29-5.92)
2 receivers	261.149 (0.03-0.65)	625.171 (0.07-1.57)	2.532.199 (0.32-6.40)
Cost per Receiver	(0.01-0.32)	(0.03-0.70)	(0.15-3.20)
10 receivers	808.893 (0.10-2.04)	1.494.035 (0.19-3.78)	4.043.303 (0.50-10.22)
Cost per Receiver	(0.008-0.20)	(0.01-0.37)	(0.04-1.01)

  - The *factory deploy()* and the *createDelivery()* functions are considerably more expensive than the *accept()* and *finish()* functions. An exception to this statement occurs when we use long keys and the number of receivers is big.
  - The *factory deploy()* is executed only once and its cost is amortized by the number of deliveries created using the factory. We have to take into account that the cost of the two more expensive operations (*factory deploy* and *createDelivery* functions) is distributed between the owner of the service (the *Factory* deployer) and the sender of the delivery.
  - The only function executed by the receiver, *accept()* is very cheap in terms of gas.

- Number of receivers.
  - The *factory deploy()* and the *accept()* functions are independent of the number of receivers.
  - The *createDelivery()* and the *finish()* functions depend on the number of receivers and the total cost grows as the number of receivers increases. However, although the total cost grows, the cost per receiver decreases, making the multiparty protocol more efficient with a bigger number of receivers.

- Length of the security elements.
  - The most costly operation, *factory deploy()* function, is independent of the length of the parameters, allowing us to use longer parameters.
  - The cost of execution of the *createDelivery()* and the *accept()* functions rises slightly with the length of the parameters.
  - *finish()* is the function that includes the verification of the ZKP and is the function more affected by the change in the length of the parameters. However, as the number of receivers increases, the cost introduced by the use of longer parameters is reduced.

- Execution delay
  - The execution delay is not greater than 900 ms in any case. This means that the execution time is small compared with the block validation time in the Ethereum network through the mining process, so this will be the greatest component of the total delay value.
  - The validation delays are closely related to the gas cost used in each function. This way a user can choose the gas cost values, analyzed above, that best adjust to its cost and delay requirements That is, the delay and the cost of the notification can be controlled by adjusting the gas price.
  - As it could be expected, the greatest execution time corresponds to the *finish()* function, where the ZKP is checked while *accept()* is the quickest function. In addition, longer parameters require bigger execution times.

10.6 Conclusions

It has been proved that blockchain-based technologies can be very useful in the design of a qualified certified notifications service, solving the problems related to the use of TTPs.

The results of the analysis of properties included in section 10.4 prove that the protocol achieves the ideal properties of the service: effectiveness, fairness, timeliness, non-repudiation and confidentiality, as it is summarized in a table that compares the new protocol with previous studies Table 10.5.

Table 10.5: Comparison of the properties of confidential multiparty certified notifications protocol without TTP with previous proposals

Property	**Non-Confidential Protocol**	**Confidential Protocol With TTP**	**Confidential Protocol Without TTP**
Effectiveness	YES	YES	YES
Fairness	STRONG	WEAK	STRONG
Timeliness	YES	YES	YES
Non-repudiation	YES	YES	YES
Confidentiality	NO	YES	YES
Evidence Transferibility	YES	NO	YES
Use of TTP	NO	OPTIMISTIC	NO

In Chapter 9 we proposed two protocols to send multiparty certified notifications also based on blockchain technology. However, with respect to the scheme we propose here, the first protocol in Chapter 9 does not have the property of confidentiality, since the content of the notification is public. Although the second certified notifications proposal in Chapter 9 achieves confidentiality, in order to ensure the fairness of the exchange, it needs the intervention of an independent Trusted Third Party (TTP) that takes actions as an intermediary to resolve disputes between sender and receivers. Thus, the proposed scheme in this chapter improves the previous systems because it achieves confidentiality with respect to the former proposal in Chapter 9, and it removes the intervention of the TTP with respect to the latter proposal in Chapter 9.

In addition to that, our new scheme not only provides fairness and confidentiality without needing the intervention of a TTP thanks to the implementation of a ZKP primitive inside the protocol but also the fairness that guarantees is *strong*. In Chapter 9 the protocol that has confidentiality has some cases where users (sender and receivers) can have contradictory evidence. That is, for example, a malicious sender can obtain, at the same time, a piece of non-repudiation evidence from a receiver and cancellation evidence from the smart contract. Thus, the sender can get evidence that a certain notification has been canceled or has been finished successfully. That is the reason why the fairness of the previous proposal is *weak*, in contrast to the *strong* fairness of the new proposal of this chapter.

Summarizing the comparison of features included in Table 10.5 we can say that the presented protocol not only allows the protocol to be executed without the involvement of a TTP but also improves the behavior of the previous confidential protocol offering strong fairness and transferability of evidence. These properties were only achieved by a non-confidential protocol.

Regarding the performance of the proposals, we can say that the confidential protocol without TTP executes more on-chain operations that the confidential protocol with TTP. For this reason, the gas cost could be bigger. However, we have to take into account that this proposal does not require payment for the services of the TTP, so the gas cost has to be added to the cost of the TTP for the previous protocol. Moreover, the new protocol obtains greater benefits of the multiparty operation, since the previous protocol executes independent resolutions of the exchange for each receiver.

Table 10.6 shows the differences in the cost per receiver between the confidential protocol with TTP and the new confidential protocol without TTP, for the function that differs the most between the two protocols, *finish()* function. In this comparison, we have to take into account two important facts. The first one in that in the protocol with TTP the *finish()* function is not mandatory. The second one is that while in the protocol without TTP the *finish()* function is executed once, independently of the number of receivers, in the protocol with TTP the function is executed once for each receiver. Thus, the cost for the protocol without TTP decreases with the number of receivers.

Table 10.6: finish() function gas cost and equivalent US-Dollars price with 1 Gwei - 20 Gwei gas price

Cost per receiver	**Without TTP** 256 bits	**With TTP**
1 receiver	(0.02-0.49)	(0.019-0.39)+TTP cost
2 receivers	(0.01-0.32)	(0.019-0.39)+TTP cost
10 receivers	(0.008-0.2)	(0.019-0.39)+TTP cost

We also have evaluated the delay in the execution of the functions in both protocols. For the confidential protocol without TTP, after the execution of the functions, the block including the transaction has to be confirmed in the blockchain. The execution time for all the functions is lower than 900 ms while the block confirmation delay depends on the gas price but can be included in the interval of 15-30 seconds, being the main component of the delay of the protocol. This delay can be considered much lower than the delay introduced by the resolution of an exchange executed by a TTP. When a user asks the TTP for a resolution, the TTP has to queue the request, check the received data and make a decision about the final result of the exchange, so the user could have to wait a considerable amount of time to obtain a response, compared with the block confirmation time.

To conclude, this chapter proposes a new protocol that achieves the fulfillment of all the desired properties of a certified notifications service using blockchain. Since now, these properties have been partially achieved by the existing protocols. So the new scheme includes the best part of each one of the previous protocols in a single protocol. This way, the new protocol allows the certified notification of confidential data without the need for a Trusted Third Party. The new protocol has been detailed, implemented and analyzed, obtaining the following conclusions:

- The presented protocol uses blockchain to allow a fair exchange for certified notifications. This way, data, proof of origin and proof of reception are exchanged without the involvement of any Trusted Third Party while achieving the desired properties for fair exchange protocols: efficiency, strong fairness, transferability,

non-repudiation and confidentiality. For this reason, the protocol is an obvious improvement from previous proposals.

- The protocol allows multiparty deliveries, useful to reduce costs in those scenarios in which the same delivery involves several receivers, like notifications for enterprise staff, shareholders meetings,...

- The delivered data is confidential, only the sender and receiver can access the data. This means that the smart contract must manage the exchange and ensure fairness without being able to access the delivered data. In protocol terms, this means that the delivered data must be encrypted until acceptance when the Non-Repudiation of Reception is provided, and the smart contract, without having access to the decryption key, must ensure that all the receivers will be able to decrypt the same message.

- To achieve all the desired properties, including confidentiality, the protocol uses more complex cryptography than previous protocols. The protocol has been implemented and tested on Ethereum to analyze the performance of these operations. The results show that although the execution costs are slightly greater than those of non-confidential protocols, the protocol is viable. The smart contracts are described in this chapter.

- With the introduction of this new protocol, users can choose between protocols that register the contents of the delivered data or the confidential protocol. Due to execution costs, users must use the non-confidential protocol not only when the data registry in the blockchain is required, but also in all the deliveries where confidentiality is not required.

- The performance analysis shows that the execution costs depend on the number of receivers, the gas price and the length of the cryptographic parameters. The efficiency of the protocol rises with the number of receivers. The gas price will affect the execution delay, but since the deliveries are asynchronous the execution delay is not a critical parameter. We have also evaluated the execution delay but the resulting values are small enough compared to the block verification time, so the total delay depends strongly on the block verification time. The length of the cryptographic parameters are related to the strength of the confidentiality but they don't have any effect on the strength of the fairness offered by the protocol.

The performance analysis has allowed the detection of the most costly operations. The use of longer parameters leads to higher levels of protection but is expensive in terms of cost. As a future work, we intend to implement the protocol using Elliptic Curve Cryptography. According to our initial tests, we expect to have similar results to those obtained with lengths of 256 bits while offering higher security standards. Thus, the Table 10.6 could have the appropriate parameters to establish a comparison point for future works.

As a significant achievement in the development of this protocol, we have secured a European patent titled *"Method for notifications and certified deliveries based on blockchain technology"*, with Patent number ES2802420. This patent was officially

issued on April 25, 2022. For more details, the full documentation is available for consultation at consultas2.oepm.es.

CHAPTER 11

IMPROVING THE EFFICIENCY OF A CONFIDENTIAL MULTIPARTY CERTIFIED NOTIFICATIONS PROTOCOL

In Chapter 8 and Chapter 9, we have discussed blockchain-based methods for certified notification services, where users receive the sender's proof of message dispatch and electronic verification of delivery attempts, that is, Non-Repudiation of Origin (NRO) evidence against Non-Repudiation of Reception (NRR) evidence. The protocol we presented in Chapter 10 builds upon these discussions by achieving the best properties of the solution presented in Chapter 9. It operates without the involvement of a TTP at any stage, enabling the certified notification of confidential data.

To ensure confidentiality in a public smart contract environment, the protocol forces that data remain encrypted until the receiver accepts it, at which point the Non-Repudiation of Reception (NRR) proof is issued. Importantly, the smart contract does not access the decryption key directly, ensuring it cannot be embedded in the transaction in clear text. The protocol is designed to guarantee that receivers can decrypt the data upon accepting delivery, and in multiparty scenarios, it ensures that all parties decrypt the same data.

In this chapter, we will explore significant improvements to this protocol, focusing on enhancing its efficiency and scalability while maintaining security and privacy. These enhancements are critical as they address potential bottlenecks associated with blockchain technology, such as transaction speed and cost, which are pivotal for real-world applications.

11.1 Contribution

Even though the confidential multiparty certified notifications protocol without Trusted Third Party explained in Chapter 10 achieves the desired properties, and the chapter presents a performance analysis proving its viability, the efficiency of the system would

be improved if the costs associated with the deployment and execution of the smart contracts could be reduced. In this chapter we present how the costs can be reduced by acting in three different aspects of the protocol: the storage system for the encrypted data, the cryptographic algorithms used and the method used to create the smart contracts for each delivery. The resulting protocol maintains the desired properties of the original one since the steps of the fair exchange and the generated proofs are the same. We include in this chapter the description of the improved protocol, the cost analysis and the comparison of the results with those of Chapter 10, proving how the execution costs have been reduced significantly.

This improvement also uses the Ethereum blockchain network due to its main features, providing a scenario to compare the implementation of both protocols. Then, we have changed the storage system for the encrypted data, replacing the on-chain storage of the encrypted certified notification content with the use of the decentralized storage system IPFS (see subsection 4.9.1). Moreover, in order to be able to use shorter encryption keys to maintain the security level, we have changed the *ElGamal* cryptosystem algorithm for Elliptic Curve Cryptography (ECC), described on subsection 4.7.3. Finally, to obtain the best price rates on new certified notification smart contracts deployment, the new implementation uses the Factory Clone programming pattern, described on subsection 4.8.1.

11.2 Protocol design

The improvement of the original Certified Notifications Protocol (Chapter 10) presents the same phases of the original version, with some enhancements thanks to the new technologies introduced. There are three compulsory phases *Creation*, *Acceptation* and *Finalization*, and two optional phases, *Cancellation* and *Verification*.

However, prior to explaining each phase in detail, we describe how the protocol works in general terms. Alice, the certified notification sender, registers the delivery encrypted message, achieving the necessary confidentiality. Nevertheless, the smart contract will have to check that all receivers can decrypt the same delivered message. Before this verification, the receivers have to generate the called *Notification Keys* and accept the certified notification in the *Accept* phase. The *Notification Keys* will be finally used by Alice when she executes the *Finish* phase, to encrypt the delivery message encryption key, allowing the user to obtain the notification content. The *Accept* and *Finalization* phases are restricted by two timelines defined by Alice. A first deadline, *term1*, sets the time before which the receivers can accept the notification. From this (*term1*) to the next deadline, (*term2*), Alice can finish the certified notification, sending the encryption key to the receivers that have accepted it.

The verification made by the smart contract by means of a ZKP (Schnorr Zero-Knowledge Proof [46]) is performed on-chain, assuring that Alice provides the same notification content for all receivers, preserving, at the same time, the confidentiality of the exchanged message. Specifically, the ZKP doesn't allow the smart contract to obtain the notification content, however, it provides the tools to detect that the key to obtaining the decrypted message is the same for all receivers.

The phases of the original protocol have been modified as follows:

Creation

$Alice$ generates her pair of public and private encryption keys (a, A) and generates the random seed v that will be used to encrypt the message M. If it is necessary, for this encryption operation, the content of the message can be divided into fragments ($M[j]$).

Then, before the encryption, $Alice$ generates the element V of an adapted method of the Elliptic Curve Integrated Encryption Scheme (ECIES) [173], [39]. Element V is used as a commitment to the emission of the right key by *Alice*. It will be verified by the smart contract through the non-interactive Schnorr Zero-Knowledge Proof [46], used on the *Finish* phase to verify that all receivers have access to the same v, to decrypt the registered message.

Subprotocol 5: Step 1. Creation

1. $Alice$:
   Generation of $a \leftarrow [1, n-1]$, $A = G\text{x}[a]$, $v \leftarrow [1, n-1]$, M
   $key = random.seed(hash(v))$
   Encryption of M.
   If required, fragmentation of M in blocks: $M[j]$
   $V = G\text{x}[v]$
   FOR $j = 1$ **TO** $M.length$
   $C[j] = M[j] \oplus key$
   $key = random.seed(hash(key))$
2. $Alice \blacktriangleright$ IPFS: Upload C = $hashIPFS$
3. $Alice \blacktriangleright SM.creation(Alice, B, V, hashIPFS, A, term1, term2, D)$
4. SM:
   $State_i = Created, \forall i$

Following the encryption of the message, the improved implementation performs the upload of the encrypted message to the IPFS system, from which $Alice$ obtains the content CID (i.e. IPFS identification label of a message) of the delivery. At this point, $Alice$ executes the function *creation()*, where the Factory Clone smart contract clones a new Delivery smart contract, with all the parameters set. This function includes: the set of receiving B, $Alice$ commitment V, the IPFS CID of the encrypted delivery $hashIPFS$ and two deadlines: $term1$ for each receiver to accept the delivery and $term2$ for $Alice$ to finish the exchange. Also, as an additional parameter, there is an optional payment for the service or a deposit D. Finally, the state is updated for each receiver i and the Smart Contract defines the state as *Created*.

Accept

If a receiver decides to accept a certified notification, he has to do it before the deadline $term1$, by executing the corresponding function $accept()$ of the Delivery smart contract. Otherwise, a rejection of the receiver is assumed. In the *Accept* phase, the receiver Bob_i, generates its own public and private shared notification keys (b_i, B_i), which will be shared between a concrete recipient (Bob_i) and the sender. To share the notification key that will be used in the $Finish$ phase on the ZKP step, first Bob_i

generates his encryption nonce s_i and the challenge variable c_i. Next, he generates parameters Z_{i_1}, Z_{i_2} to allow Alice to get the secret shared key b_i.

When Bob_i has generated all the required parameters, he can execute the Smart Contract function $accept()$, sending Z_{i_1}, Z_{i_2}, B_i and c_i parameters, which are needed by Alice on the Finish phase to generate the response (r_i) to the challenge (c_i) sent by Bob_i to $Alice$.

Subprotocol 6: Step 2. Accept

1. Bob_i :
   Generation of the parameters: $b_i \leftarrow [1, n-1], B_i = Gx[b_i], s_i$
   $Z_{i_1} = Gx[s_i], Z_{i_2} = (Ax[s_i]) \oplus (b_i)$, $c_i \leftarrow [1, n-1]$
2. $Bob_i \blacktriangleright SM.accept(Z_{i_1}, Z_{i_2}, B_i, c_i)$
3. SM:
   IF$(now < term_1)$ AND $(Id == B_i)$ AND $(State_i == Created)$
      $State_i = Accepted$
      Add B_i to B'

Finish

Between the deadlines ($term1$, $term2$), $Alice$ can finish the certified notification process executing the $finish()$ function for all Bob_i that have accepted the notification, from now on Bob' users.

First of all, in this phase, $Alice$ must create the response for each challenge received by Bob' users, using the element v, used in the first phase of the protocol to encrypt the delivery, the secret shared notification key (b_i) and Bob_i challenge (c_i). So, $Alice$ must obtain from the delivery smart contract B_i public key, Z_{i_1} and Z_{i_2} parameters and the receiver challenge c_i. Using these parameters, $Alice$ decrypts the shared secret notification key (b_i) and then encrypts the delivery encryption key v using the challenge received and Bob_i shared secret key. Finally, she invokes the $finish()$ function of the smart contract to submit the result.

Then, Bob_i can request the parameters needed for the smart contract, getting the IPFS CID of the delivery ($hashIPFS$), Alice's public key A and Alice challenge response r_i. Then, Bob_i needs to obtain the encrypted delivery, through an IPFS request sending the $hashIPFS$ parameter. Once, Bob_i has the encrypted delivery message C, using the received r_i, his c_i and b_i parameters only have to decrypt C through an XOR function with the key derived from v.

Cancellation of Acceptance

The protocol takes into account the possible case in which the sender does not finish the certified notification. Therefore, an optional subprotocol is provided that will be executed in case $Alice$ does not provide the decryption key once a receiver has accepted the certified notification.

A user B_i is allowed to cancel if $term2$ has been exceeded and the delivery status is *Accepted*. The smart contract will change the state of the delivery from *Accepted* to *Canceled*.

Subprotocol 7: Step 3. Finish

1. $Alice$ ▶ SM.getParameters(): $Z_{i_1}, Z_{i_2}, B_i, c_i$
2. $Alice$:
   Decrypts: $b_i = Z_{i_2} \oplus (Z_{i_1}\text{x}[a])$
   Generation of r_i=$v - b_i * c_{i\ mod n}$
3. A ▶ SM.finish(r_i)
4. SM:
   IF ($Id == Alice$) AND (($term_1 < now < term_2$) OR (($Bob' == Bob$) AND ($now < term2$)))
      FOR ($\forall Bob_i \in Bob'$)
         IF $V == G\text{x}[r_i] + B_i\text{x}[c_i]$
            $State_i = Finished$
      FOR ($\forall Bob_i \notin Bob'$)
         $State_i = Rejected$
      Deposit D is refunded to $Alice$
5. Bob_i ▶ SM.getParameters(): $hashIPFS, A, r_i$
6. Bob_i ▶ IPFS: Download $hashIPFS = C$
7. Bob_i: $v = r_i + b_i$*$c_{i\ mod n}$
   $key = random.seed(hash(v))$
   FOR $j = 1$ **TO** n
      $M[j] = C[j] \oplus key$
      $key = random.seed(hash(key))$

Subprotocol 8: Cancellation of Acceptance

1. B_i ▶ SM.cancel()
2. SM:
   IF($now >= term2$, $Id == B_i$ AND $State_i == Accepted$)
      $State_i = Canceled$

Verification

As in the original implementation, all variables and states stored on the smart contract are public, allowing any verification carried out by the sender $Alice$, any receiver Bob_i of the delivery and, also, from any third party.

11.3 Implementation

The implementation of the improved protocol can be found in a GitHub repository[1] maintained by the Security and e-Commerce (SECOM) Research Group from the University of the Balearic Islands. It has been developed from the creation of the smart contracts *Factory Clone* and *eDelivery* (that represents the certified notification) that provide the methods explained in the previous section. The *eDelivery* smart contract is responsible for the execution of the ZKP using an Elliptic Curve Cryptography (ECC)

[1] https://github.com/secomuib/ConfidentialRegisteredEDeliveryProtocolImproved-IPFS-ECC

library that provides the arithmetic operations over elliptic curves to achieve the ZKP calculations. For this purpose, the smart contract implementation uses the Witnet Foundation Elliptic Curve Library[2]. The *Factory Clone* smart contract has the *clone()* method, which follows the Open Zeppelin library[3].

The two smart contracts of the improved implementation contain the same functions as the two smart contracts of the original implementation. There are only some changes due to the cryptographic improvement (i.e. from *ElGamal* cryptosystem to ECC). An explanation of the Solidity code of the smart contract is omitted due to the space limitation of the thesis. However, the full content of our implementation is available at the GitHub site previously specified.

Besides the Solidity smart contracts code, we have also implemented a front-end interface using JavaScript React that allows the execution of the protocol in a user-friendly mode. Moreover, any user can make use of this interface to upload and download the encrypted messages of the delivery to the IPFS system.

In addition to that, the user interface has to be able to encrypt and decrypt the delivered messages and generate all the elliptic curve parameters. In order to compute the parameters, we use an elliptic curve Javascript library[4] configured with the same *secp256k1* curve as in the Solidity library.

11.4 Performance analysis

Once we have finished the improved implementation of the protocol, we have tested it together with the original protocol, in order to determine the efficiency improvement in terms of cost. In the test, we have evaluated the methods of the smart contract that define the three compulsory phases *Creation*, *Acceptation* and *Finalization*. These functions are the ones that execute more instructions on-chain and they depend on the three new technologies introduced. The cost tests have been made using the Hardhat environment, specifically executed over the provided local blockchain node.

In order to obtain data to determine the improvements introduced by the use of the IPFS system, the cost tests of the original implementation have been carried out again to have results for both protocols with a different number of receivers and different lengths of the delivered messages.

The improved implementation, as it has been explained in previous sections, uses ECC. Specifically, we have selected the *secp256k1* curve. This change has improved significantly the encryption security of the implementation, compared to the security provided by the same key length of the cryptosystem *ElGamal*. In order to analyze and compare the costs of the two implementations we have selected the key lengths which provide the same security level, and performed the required experiments to obtain the costs. Following the NIST recommendations [174], we have selected the key of 3072 bits for *ElGamal*, which provides the same security level of keys of 256 bits in the ECC implementation. Therefore, with this change, we have highly reduced the key bits stored in the blockchain.

[2] https://www.npmjs.com/package/elliptic-curve-solidity
[3] https://docs.openzeppelin.com/contracts/4.x/api/proxy#Clones
[4] https://www.npmjs.com/package/elliptic

We have evaluated the gas cost of the functions for both the original and the improved protocol, taking into account the parameters that affect the cost.

- The *createDelivery()* function of the original protocol requires an amount of gas that depends on the message length. For this reason, we have tested it with three message lengths (10, 5000 and 10000 characters). In contrast, the message length doesn't affect the improved protocol, because it only stores on the blockchain the CID of IPFS that identifies the message.
- The *accept()* and *finish()* functions are independent of the message length, but *finish()* depends on the number of receivers.
- The *deploy()* function of the Factory Clone smart contract doesn't have significant changes from the original implementation and it is independent of the number of receivers and the message length. It costs 3,419,964 Gwei in both protocols.

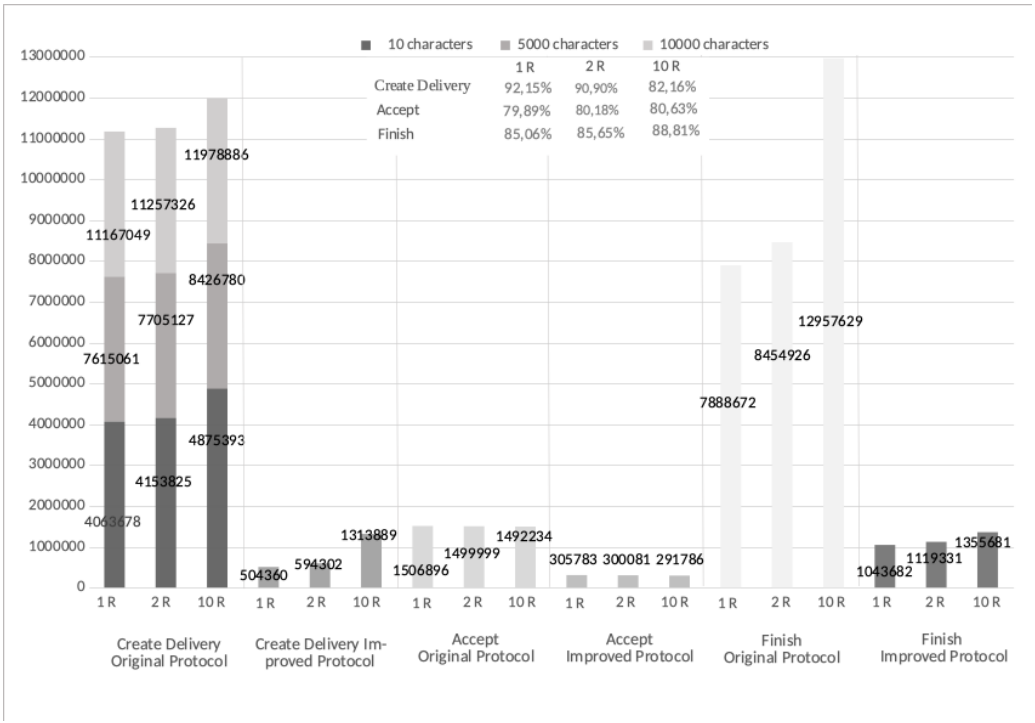


Figure 11.1: Gas cost of the improved confidential multiparty certified notifications smart contract

Figure 11.1 shows the results of the analysis. The horizontal axis of the graph is divided into the three compulsory phases (*Create Delivery*, *Accept* and *Finish*). Each one of these parts is also divided into two sections, one for the *Original Protocol* and the other one for the *Improved Protocol*.

Sections are formed by three bars that represent the test results of a certified notification with 1, 2 and 10 receivers, (1R, 2R and 10R, respectively). Over each vertical bar, the cost results of the corresponding function test are indicated, computed in values of gas, setting a gas price of 1 gwei.

The first section of Figure 11.1 represents the *Create Delivery* of the *Original Protocol*. In this section, there are three bars that represent the gas cost for 1, 2 and 10 receivers. In addition, each bar also represents the results for the three message lengths, presented in the legend at the top of the graph.

Finally, in the center of the figure, we present the percentage cost reduction between the results of the *Improved Protocol* and the *Original Protocol* of the same method, providing a quantitative evaluation of the improvement. The percentage of improvement for the *Create Delivery* function has been computed from the average results of the three message lengths for each number of receivers.

From the results introduced in Figure 11.1 we can determine how each improvement has affected the efficiency of the protocol.

- **Elliptic Curve Cryptography**
  - The usage of the Elliptic Curve Cryptography provides a significantly important enhancement in gas cost thanks to the short keys of 256 bits in front of the 3072 bits keys of the *ElGamal* cryptosystem to achieve the same security level. Specifically on the *createDelivery()* and *finish()* functions, as it can be seen in Figure 11.1.
  - In the results obtained from the original implementation with 3072 bits of key length, *accept()* and *factory deploy()* functions are the cheapest functions, followed by *createDelivery()* and *finish()*, the ones that use the 3072 bit keys of *ElGamal* cryptosystem.
  - In the improved implementation, the cheapest functions are *accept()* and then *createDelivery()*, followed by *finish()* and *factory deploy()*. This is the same order obtained when testing the original implementation with shorter keys (256, 512 or 1024), as it is presented in Chapter 10.
- **Factory Clone**
  - The improvement generated by the use of the Factory Clone is represented in the gas costs of the *createDelivery()* function. A large part of the improvement percentage of the *createDelivery()* section in Figure 11.1 is achieved thanks to the introduction of this feature.
  - We have also analyzed the gas costs of the functions using the Factory Clone vs. an implementation without this programming pattern in order to evaluate the improvement when this change is isolated from the others. The results show that function *createDelivery()* obtains a 71,78%, 68,36% and 49,45% improvement for 1, 2 and 10 receivers respectively.
- **Use of IPFS**
  - The *createDelivery()* function has also obtained a cost reduction thanks to the use of the IPFS system to store encrypted certified notification messages. As it is represented in the section *Create Delivery* of Figure 11.1, this improvement increments with the number of characters used.
  - This feature enhances the implementation by providing an easy way to deliver not only text messages but also any type of file and size.

When the three improvements are evaluated together, we have obtained a cost reduction between 79,89% and 92,15%, as it can be seen in the table that appears in Figure 11.1. From this cost analysis, we can state that we have maintained the most important properties of the certified notification system on the improved protocol, and we have significantly boosted the efficiency in terms of cost, related to the costs obtained in Chapter 10.

11.5 Conclusions

There are some important features that will help to introduce blockchain-based applications to the general public. We think that improving efficiency and costs is capital for such adoption. In this chapter, we present how we have improved a powerful blockchain-based security protocol for certified notifications that originally was able to remove the need for Trusted Third Partys while providing fairness, confidentiality and multiparty capabilities. Since the original protocol fulfilled the desired security properties we have focused our improvement on the efficiency and cost of the protocol, modifying the interactions among the actors of the protocol and also changing the cryptographic operations performed in some phases of the protocol. Concretely, we have changed the place where the encrypted data are stored, we have also changed the cryptosystem used to encrypt the data and we have used more efficient functions in the smart contracts.

With these improvements, we have achieved a solution where the encrypted data can be accessed off-chain. Moreover, the keys used in the new cryptosystem can be shorter while maintaining the same security level. We have described the new protocol and how we have implemented it. Later the original and the improved protocol have been compared. We have shown how the new protocol reduces significantly the cost associated with the deployment and execution of smart contracts. We have presented the percentage of reduction for each function depending on the number of receivers. For all functions and number of receivers, the improved protocol presents a cost reduction higher than 80%. We can conclude that the presented protocol is secure and efficient, making it a perfect solution for the certified notifications service. Thus, the patterns that are presented in this chapter can be shown as a set of best implementation practices for security protocols deployment using blockchain technology.

CHAPTER 12

TWO-STEPS CERTIFIED NOTIFICATIONS PROTOCOL

In this chapter, we explore a new approach by integrating Soulbound tokens (SBTs) with Rejectable Non-Fungible Tokens (RejNFTs) and Identity-Based Encryption (IBE). As detailed in subsection 4.6.2, SBTs are a type of Non-Fungible Token (NFT) that is permanently owned by a single individual and cannot be sold or transferred. On the other hand, in Chapter 7 we presented a protocol enhancement to the ERC-721 standard, which includes the possibility of rejection by the recipient, thus enabling selective reception of NFTs. This integration leads to the development of Rejectable Soulbound tokens (RejSBTs), which combines the properties of the SBTs and the RejNFTs. This functionality is particularly useful for representing certified notifications as RejSBTs, which effectively track the distribution and confirmation of notifications, providing reliable evidence of Non-Repudiation of Origins (NROs) and Non-Repudiation of Receptions (NRRs).

This protocol presents a significant improvement in the management of certified notifications over previous methods discussed in earlier chapters. By streamlining the process to only two key actions, issuance and acceptance of a RejSBT, we simplify the interaction needed to manage notifications. Once a RejSBT is accepted, the new holder can apply Identity-Based Cryptography, described in subsection 4.7.4, to decrypt sensitive data embedded within the token, with the help of a Middleware service. This protocol is designed to ensure the confidentiality of the notification content and to prevent the receiver from accessing the message until they have accepted the notification.

12.1 Contribution

The objective of the protocol presented in this chapter is to enhance the efficiency and security of managing electronic certified notifications. Traditional certified notification protocols typically involve a three-step exchange: first, the sender dispatches an en-

crypted message; next, the receiver acknowledges and accepts the message; and finally, the sender provides the decryption key or method, enabling the receiver to access the message's content. This multi-step process, while secure, often results in delays and operational inefficiencies, particularly in scenarios requiring rapid confirmation and response. The new protocol proposed herein simplifies this procedure by reducing the interaction to a two-step process, thanks to the combination of Soulbound tokens and Rejectable Non-Fungible Tokens. This advancement marks a substantial improvement over existing methods by minimizing the necessary interactions between the sender and the receiver and introducing an integrated approach to encryption and decryption based Identity-Based Encryption (IBE).

Regarding Blockchain technology, one of the most widespread uses of Ethereum or compatible blockchain networks is the use of tokens, such as the ERC-20 and ERC-721 token standards. The first one represents fungible tokens, interchangeable between them because they all represent the same value, and the second one represents a Non-Fungible Tokens (NFTs), unique and not interchangeable. Since these unique tokens can be transferred and the transfers are recorded on the blockchain, they could be considered a valid medium of information transfer. On the other hand, Soulbound tokens are a kind of NFT specially designed to avoid their transfer after the initial assignment.

The Soulbound tokens can be very useful in representing a certified notification sent from a user to a specific receiver. Due to the notification is unique as an NFT, and with the non-transferable property, the receiver won't be able to transfer this token to another user. This means that there can only be one transfer, the initial transfer from the sender to the receiver.

However, if we want to use the Soulbound tokens as a representation of a certified notification, this standard would lack an important property: the impossibility of rejecting the reception of a token. This problem is because we can transfer tokens defined by the standards cited before, but we cannot perform a selective reception, because we cannot reject these transfers.

This can be solved with a rejectable token, that enables the selective reception of tokens, that is, allowing the receiver to reject its transfer. In our protocol described in Chapter 7, we proposed an improvement to the NFTs standard defined in ERC-721 to achieve selective receipt, in the form of Rejectable Non-Fungible Tokens (RejNFTs). This kind of token allows selective reception, implementing the possibility to reject them. This protocol can also be applied to SBTs to obtain a new standard defined in this chapter, Rejectable Soulbound token (RejSBT). In this RejSBT standard, we also have included a deadline to let receivers accept or reject the transfer proposal.

Once the receiver has accepted the certified notification, represented by a Rejectable Soulbound token, we can use a Middleware service that can check the ownership of the notification, and, using Identity-Based Cryptography, decrypt the confidential information stored in that service and show it to the receiver.

In this chapter, we present an improvement of the certified notifications protocol that leverages the newly defined RejSBT. The protocol ensures the certified notification or delivery of data providing both NRO and NRR proofs. Moreover, the use of Identity-Based Cryptography allows to achieve the property of confidentiality of the delivered notification and at the same time hides the contents of the notification to the receiver until he accepts the notification.

This proposed new protocol for confidential certified notifications improves the previous protocols presented in this thesis, as it requires the involvement of the sender and the receiver only once. That is, the exchange is performed in two steps: the sender sends a RejSBT that represents the notification, and the receiver accepts the RejSBT. Once the receiver has accepted the RejSBT, the middleware delivers the message to the receiver, decrypting it using Identity-Based Cryptography.

For all the reasons explained above, we can underline the novelty of the proposal compared with other recent studies. The protocol presented in this chapter uses tokens to represent notifications, allowing the protocol to benefit from the intrinsic properties of tokens. One important aspect of the novelty of the protocol is the proposition of a new kind of token, the RejSBTs. This new kind of token is suitable for the application described in this chapter but will also be useful in future works to solve the challenges in the definition of other protocols. The adoption of the RejSFTs in the definition of a protocol for confidential certified notifications allows the achievement of a goal in this kind of protocol, to reduce the number of steps to be executed by the involved actors reducing them to two, the minimum number until the date, one step per actor. This feature in itself represents an important novelty.

To summarize, the contribution of the protocol presented in this chapter is:

- Blockchain-based protocol for confidential certified notifications with the minimum number of steps. In the protocol the actors are only involved in one step, reducing the total number of steps to two.
- Extension of RejNFTs and definition of RejSBTs. In Chapter 7, we described the concept of RejNFT which has now been improved to enhance its functionality by adding temporal parameters.
- Use of RejSBTs and Identity-Based Encryption (IBE) to define the protocol that achieves all the desired properties of confidential certified notifications.
- Security and performance analysis of the proposed protocol.

12.2 Protocol design

In this chapter, we present a new protocol to send certified notifications used to communicate a message from a sender S to a receiver R in a confidential way. For instance, the notification can be an official communication from an *Authority* to officially inform the receiver of something important (e.g., the result of a medical test).

The protocol described in this chapter is composed of two subprotocols, *Notification Release* and *Notification Reception*. The notification is bonded to a *RejSBT* in such a way that the ownership of the token by the receiver indicates the reception of the message (i.e., the receiver's ability to read the content of the message). The ownership of the token can be publicly checked, as well as the address from whom the token was minted and transferred. Therefore, Non-Repudiation of Origin and Non-Repudiation of Reception are guaranteed.

We use the notation in Table 12.1 to describe the protocol. Additionally to the sender S and receiver R actors, we also have defined a service provider that deploys a smart contract to manage the tokens and a MW service to generate the security keys

Table 12.1: Notation of the two-steps certified notifications protocol

Name	Description
S	Notification Sender
R	Notification Receiver
MW	Middleware
$H()$	Secure hash algorithm
m	Notification message content
C	Encrypted message content
ID	Notification Identifier
K	Symmetric encryption key
d	deadline
k	Symmetric encryption key encrypted
$AES.decrypt_K(C)$	Symmetric decryption of C using K
$AES.encrypt_K(m)$	Symmetric encryption of m using K
$IBE.encryption_{ID}(K)$	Asymmetric encryption of K using an IBE scheme with ID as public key
SK	Security key used as the private key of an IBE cryptosystem
$IBE.extracts_{mS}(ID)$	Generation of the corresponding private key to ID of an IBE scheme using as master secret mS
$IBE.decrypt_{SK}(k)$	Asymmetric decryption of k using an IBE scheme with SK as private key
$U \blacktriangleright_{Blockchain}: f()$	User U call function f of a deployed smart contract
$U \Rightarrow_{SecCh} V : M$	User U sends M to V through a secure channel $SecCh$

for the notification receivers. The MW acts as the PKG of an IBE cryptosystem and publishes the public parameters to allow users to perform the encryption operations. Also, prior to the sending of any notification, the sender must ask permission to the service provider to mint the tokens.

We introduce a new approach to this kind of protocol by the tokenization of the sending together with the use of blockchain technology to attest to the transfer of the token. Then, the communication will be between two blockchain addresses, which represent the communicating parties (i.e., from the *sender*'s address to the *receiver*'s address). In addition to that, an IBE cryptosystem is used to avoid the costly PKI management and to make it easier to create new public keys for each new notification. Thus, as a requirement of any IBE schema, a PKG has to be introduced in the protocol to publish the public parameters of the cryptosystem and to create new private keys following the acceptance of the notification from a receiver.

The tokenization process needs the deployment of the new token type that represents each notification sent. For this purpose, we have defined a new type of token derived from the *SBTs*[1] and the *RejNFT*[2]. We need a combination of the features of these two types of tokens in the protocol because:

- The notification must be a non-transferable token (the reception of the notifica-

[1] Information about the motivation and the features of the *SoulBound* NFT standard can be found in: https://www.nftstandards.wtf/NFT/NFT+Soulbound+Tokens

[2] The EIP-5896 describes the motivation and the specification of the *Rejectable* tokens. Further information can be found at: https://ethereum-magicians.org/t/eip-5896-rejectable-non-fungible-token/11674

tion is symbolized by the token and it cannot be transferred in order to attest the acceptance of the message). This is a singular feature of the *SBTs*.

- The receiver can refuse the reception prior to seeing the content. This is a singular feature of the *RejNFTs*.

To implement the proposed scheme we have considered that the authority that represents the PKG is also in charge of deploying the smart contract on the blockchain for the management of the new RejSBTs and to give the right to mint tokens to the sender of the notifications.

Thus, before starting the sending of any certified notification, a previous stage must be carried out: the PKG has to issue the public parameters (IBE.$publicParameters$) of the IBE cryptosystem, as well as any intended sender has to ask permission to mint tokens. Consequently, the outline of the protocol for sending a certified electronic notification is as follows:

1. The sender (S) creates a new notification message content: m
2. S encrypts the message content m using a key K from a symmetric encryption algorithm (e.g., AES), to obtain the ciphertext C: $C = AES_K(m)$ (encrypt mode)
3. S sends C to the receiver (R) through a secure channel (e.g., an SSL connection)
4. S creates an ID of this notification and uses this identifier as a public key of the IBE cryptosystem to encrypt K: $k = IBE.encrypt_{ID}(K)$
5. The sender S mints a new token with the following information related to the certified notification: ID, H(m), H(C), H(K), k

Now, the receiver R has C and an event has been triggered from the blockchain that there is an attempt to transfer a token to its address. If R accepts the transfers, then the smart contract triggers a new event to inform the PKG that the private key corresponding to the public key ID has to be released. Note that this ID is the identifier of the notification referred by the token that has just been accepted by R.

The PKG can read the information inside the token to get the ID of the notification and then execute:

1. $privateKey$ = IBE.extract(IBE.$publicParameters$, IBE.$masterSecret$, ID)
2. Transfers the $privateKey$ to the smart contract to insert this information inside the metadata associated with the token of the correspondent notification.

This way, R can get access to the $publicKey$ and the item k from the metadata of the token that now owns.

Next, R performs:

1. K = IBE.decrypt(k, $privateKey$)
2. $N = AES_K(C)$ (decrypt mode)

Therefore, R has the message content of the notification. Note that, in the protocol description, the regular IBE naming of the key provided by the PKG ($privateKey$) is used. Despite this name, this key is not private at all in our protocol. In contrast, the protocol sends the encrypted message C through a secure channel to keep its confidentiality, thus only R can decrypt C using the $privateKey$. For this reason, we have changed the standard naming of the $privateKey$ by $securityKey$ from now on, which is more appropriate for the purpose of this element within our protocol.

In addition to that, the proposed scheme can have two different branches:

- Cancellation: The sender S can cancel the sending after minting the token and before the acceptance by R.
- Rejection: Instead of accepting the token transfer, R can reject the transfer and, thus, the PKG will not issue the $securityKey$ to enable access to the message.

Notification Release

The sending of a notification is described in Protocol 1: Notification Release. Its functionality is as follows:

1. Notification setup: the sender creates a content of the notification message m together with its identifier (ID). This identifier is a combination of the receiver's address and a timestamp. Then, S generates a secure AES key, K, and encrypts m using the symmetric cryptosystem. We denote as C the result of the encryption operation.
2. Sending the encrypted message: S sends to R the encrypted content of the notification, C, using a secure channel [175] that ensures confidentiality, integrity and replay protection, as well as mutual authentication between the entities.
3. Minting the token: the sender mints a new token bonded to the notification. To do that the sender has to link the following data to the new token: the notification identifier (ID), a newly created deadline (d), and the hash code of each of the following items: m, C, and K.

   These hash codes are computed by means of the Keccak-256 hashing algorithm. Finally, ID is used as the public key of an IBE-based cryptosystem to encrypt the AES key K getting as a result k (see description in Protocol 9 algorithm).

   The new d represents the time before which the receiver must accept the token transfer in order to have the notification content available. In addition to this, d is also the deadline for S to cancel the transfer, R to reject it, and MW to issue the key.

Notification Reception

At this point, R has received an encrypted message through a secure channel and S has minted a new RejSBT. The *mint* operation stores all parameters in the blockchain as metadata of the newly minted token and returns the *tokenId*. In addition to that, $mint$

Subprotocol 9: Notification Release

1. S creates: m identified by ID
2. S encrypts: $C = AES.encrypt_K(m)$
3. S encrypts: $k = IBE.encryption_{ID}(K)$
4. $S \Rightarrow_{SecureChannel} R : C$
5. $S \blacktriangleright_{Blockchain}: mint(ID, d, H(m), H(C), H(K), k)$

operation sets the *transferableOwners[tokenId]* mapping to the receiver address value and the state of the token is now $minted$ (See Figure 12.1). The *transferableOwners[]* mapping is used by RejSBTs to store the address of the intended new owner of the token (see section 12.3 for details). Now, it is up to R to decide if he accepts or not the token transfer before the established deadline. For example, the decision of R could be based on the verification that the hash result of the encrypted message (C) received through the secure channel is equal to the hash stored in the smart contract by the sender in the *mint* operation ($H(C)$).Thus, as illustrated in Figure 12.1 the protocol branches into two paths depending on whether R accepts or not:

resume If the receiver R accepts (Protocol 10 algorithm), then R invokes the *acceptTransfer()* function of the smart contract. Then the smart contract assigns the ownership of the token to R and the state of it is $accepted$.

a) The MW listens out for accepting transfer events from the blockchain and generates the corresponding security key (from now on SK) for the notification identifier represented by the token. Note that the key K was encrypted by S using an IBE cryptosystem with the public key ID of the notification. Therefore, the MW, which possesses the *masterSecret*, can generate and submit the related security key to the blockchain, making it available to the receiver. Once the MW has generated SK, it submits the key to the smart contract indicating the $tokenId$.

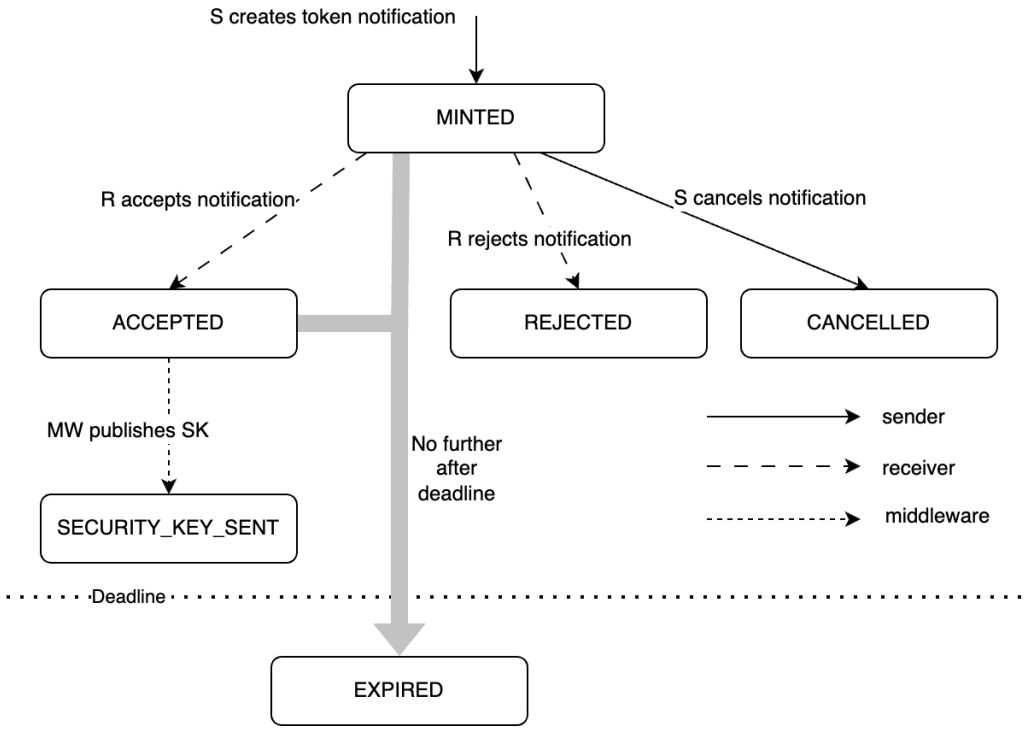
b) After receiving the data from the MW, the smart contract adds the SK as a metadata of the token identified by $tokenId$ and updates the token status to $securitykeysent$ (See Figure 12.1).

c) R can get the security key SK from the blockchain and then he computes $K = IBE.decrypt_{SK}(k)$. In section 12.4 the fairness and the confidentiality of the protocol will be analyzed. This analysis proves that the system is sound and protected from fraud attempts performed by the MW.

d) Next, R decrypts the message C received through the secure channel using the recently computed K: $m' = AES.decrypt_K(C)$. Note that, since R has accepted the transfer, R has checked that the hash code of the received encrypted message is equal to H(C) inside the token metadata. Therefore, the hash code of the decrypted message m' must match the hash code of the message stored by the sender in the *mint* operation: $H(m') = H(m)$.

resume If the receiver R does not accept, R can actively reject the transfer or can simply do nothing. In the first case, R has to call the *rejectTransfer(tokenId)* function

Figure 12.1: States of the two-steps certified notifications protocol



Subprotocol 10: Notification Reception

1. $R \blacktriangleright_{Blockchain}$: $acceptTransfer(tokenId)$
2. $MW \blacktriangleleft_{Blockchain}$: event $AcceptTransfer(from, to, tokenId)$
3. $MW \blacktriangleright_{Blockchain}$: retrieveTokenData($tokenId$)
4. MW generates security key: $SK = IBE.extracts_{masterSecret}(ID)$
5. $MW \blacktriangleright_{Blockchain}$: $sendSecurityKey(tokenId, SK)$
6. $R \blacktriangleright_{Blockchain}$: retrieveTokenData($tokenId$)
7. R computes: $K = IBE.decrypt_{SK}(k)$
8. R decrypts: $m = AES.decrypt_K(C)$.

of the smart contract before the deadline d and if the transfer has not been canceled by S, then the token state will be set to *rejected* by the smart contract. In the other case, if R has neither been accepted nor rejected and the sender has not canceled the token transference, the token state will be *expired* after the deadline.

resume If R has neither accepted nor rejected the notification, then the sender S can cancel the transfer before the deadline d. In this case, S calls the *cancelTransfer(tokenId)* function of the smart contract. This function sets the token state to *canceled* as long as the token transfer has been previously accepted or rejected by R and the current time and date are prior to the deadline (see Figure 12.1).

After the execution of this protocol, if the receiver has accepted the notification, then the following evidence has been collected:

- **Sender** S**:** the blockchain stores a signed transaction by the receiver accepting the notification, which can be easily checked by means of the smart contract that returns the notification state. Moreover, the availability of the security key, published by the MW, can be also checked.
- **Receiver** R**:** R has received C from S through a secure channel. The *mint* operation registered on the blockchain by S is non-repudiation evidence to prove that S is the actual sender of the notification identified by ID. Also, R can verify that the decrypted result of the message content, m, matches $H(m)$ stored as an element of the token metadata.

The pieces of evidence collected by the actors in this protocol are fully analyzed in section 12.4.

Identity-based encryption (IBE)

In our protocol, we have used an IBE cryptosystem (presented in subsection 4.7.4) to encrypt the content of the certified notifications. We have carefully chosen the identifier to single out a specific certified notification transmission that we have bound to a token. Therefore, the domain range of the public keys is defined by the format used to identify the sending of a certified notification. For this purpose, we have used in our implementation a JSON object made up of the receiver address and a timestamp:

```
notificationID = {
        receiver.address,
        Timestamp
      };
```

The protocol uses a timestamp to create an arbitrary object that unambiguously designates the communication of a message in the proposed system. Then, our schema transfers a RejSBT (with the identifier $notificationID$ in the metadata structure of the token) from the sender to the receiver of the notification during the mint operation, in such a way that the ownership of the token by the receiver indicates that the message content of the notification is available by this user.

Thus, we have to implement a framework for IBE-based messaging but adapted to the blockchain technology and the token infrastructure. The actors are the following: the *Private Key Generator (PKG)*, the *sender* (S) and the *receiver* (R).

The PKG is the entity that knows the *masterSecret*[3] and publishes the sharable public parameters (*IBE.publicParameters*) of the IBE cryptosystem. The PKG accepts an IBE user's private key request and, after successfully checking that this user is the intended receiver of a certified notification, returns the IBE private key to open the

[3]The *masterSecret* is generated by the PKG and, then, it is privately stored by this entity. Then, after receiving a request from a user, the PKG uses the *masterSecret* to generate the private keys for this user [176].

message content of the notification. To do that the PKG has the following cryptographic primitive:

```
privateKey = IBE.extract(
        IBE.publicParameters,
        IBE.masterSecret,
        notificationID
    );
```

The sender of a notification creates a public key of the IBE cryptosystem and encrypts a message content using:

```
encryptResult = IBE.encrypt(
        IBE.publicParameters,
        publicKey,
        message
    );
```

In our case $publicKey = notificationID$. Then, the receiver of the notification, after receiving the private key from the PKG, can decrypt the content message by using:

```
decryptResult = IBE.decrypt(
        IBE.publicParameters,
        privateKey,
        encryptResult
    );
```

Of course, if the actions are sound then $decryptResult = message$. To sum up, the standard IBE framework adapted to our schema consists of four processes:

- ***IBE.set up***: where the $IBE.publicParameters$ are created by the PKG.
- ***IBE.encrypt***: the sender encrypts a message using a public key.
- ***IBE.extract***: when the receiver accepts the token transfer, the PKG creates a private key corresponding to the public key.
- ***IBE.decrypt***: the receiver decrypts the message with the private key provided by the PKG.

In our system we have used an implementation of the Boneh-Franklin identity-based cryptosystem as described by Boneh and Franklin in [177] that is available on the *CyptID. Cross-platform Identity-Based Encryption solution* GitHub site[4].

To address the shortcomings of the traditional Public Key Infrastructure (PKI) and to simplify key management, we have introduced an IBE cryptosystem in a new certified notification scheme in combination with blockchain technology. In this way, neither the sender nor the receiver need to obtain each other's public keys to exchange messages,

[4]https://github.com/cryptid-org/getting-started#Identity-based-Cryptography

which can reduce a lot of processes for the certificate management and the transmission of any certificate notification. Because there is no need for an online lookup for a sender to obtain the receiver's public key certificate and enable the sender to send the notification in just one step. In addition, the PKG can provide new private keys for the receiver of each notification, so it is easy to update the private keys of the receivers at each sending of a new notification.

12.3 Implementation

We have written in Solidity language some Smart Contracts to implement, test, and check the performance of the protocol presented in the previous Section. The code of these smart contracts and their tests are accessible in the *rejectable-soulboundtoken-ibe*[5] GitHub repository maintained by the Security and e-Commerce (SECOM) Research Group from the University of the Balearic Islands. We have used Polygon PoS, a compatible Ethereum Virtual Machine (EVM) blockchain, to test its performance and gas costs. The EVM-compatible blockchains use programs called Smart Contract, and a distributed Turing Complete machine, to store system state changes. These EVM blockchains use their native currency, like Ether (ETH) in Ethereum or Matic in Polygon PoS, to meter and limit the costs of resources used to execute the code. In this Section, we will see the development procedure and the description of the Smart Contracts.

In this protocol, we will use a RejSBT Smart Contract to store the information related to the confidential certified notification and to track its acceptance by the receiver. Using an improvement proposal of the NFTs, introduced in Chapter 7, that allows a selective reception of these tokens, the sender will be able to cancel the transfer of this notification, and the receiver will be able to accept or reject this transfer. At the same time, we will use a deadline to determine the maximum time allowed to accept, reject, or cancel the transfer, and to let the MW send the security key. The information about the Identity-Based Cryptography system and information related to the encrypted message will also be stored in these RejSBTs. Once they are accepted by the receiver, a MW service will send the security key to the receiver to allow him to decrypt the symmetric AES session key, which will permit decrypting the notification. All these steps will be tracked by the EVM blockchain to certify the authenticity and integrity of all the information and the steps executed by all involved parties.

To explain the implementation of the RejSBTs used to send confidential certification notifications with Identity-Based Cryptography we will divide the code into the following parts:

- Soulbound token
- Rejectable Soulbound token
- Rejectable Soulbound token with a deadline
- Rejectable Soulbound token with a deadline that stores a confidential notification using IBE

[5] https://github.com/secomuib/rejectable-soulboundtoken-ibe

Soulbound token

There is no standard implementation of a SBT so far. To do this, we will get the code of an NFT (ERC-721)[28] and will remove all the $transfer$ functions and events. We will also remove all $approve$ functions because they are only needed to enable or disable approval for a third party ("operator") to transfer or manage all of the assets of a specific owner. We will also change the event $Transfer$ to two different events, $Mint$ and $Burn$, because the SBTs cannot be transferred, but they can be minted and burned.

With this, we will get the interface of an NFT or ERC-721 as presented in Listing 20, and will simplify it by removing some functions and events. In Listing 21 we can see the new interface of an SBT.

```
interface ERC721 {
    event Transfer(address indexed _from, address indexed _to, uint256 indexed
        _tokenId);

    event Approval(address indexed _owner, address indexed _approved, uint256
        indexed _tokenId);

    event ApprovalForAll(address indexed _owner, address indexed _operator, bool
        _approved);

    function balanceOf(address _owner) external view returns (uint256);

    function ownerOf(uint256 _tokenId) external view returns (address);

    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes
         data) external payable;

    function safeTransferFrom(address _from, address _to, uint256 _tokenId)
        external payable;

    function transferFrom(address _from, address _to, uint256 _tokenId) external
        payable;

    function approve(address _approved, uint256 _tokenId) external payable;

    function setApprovalForAll(address _operator, bool _approved) external;

    function getApproved(uint256 _tokenId) external view returns (address);

    function isApprovedForAll(address _owner, address _operator) external view
        returns (bool);
}
```

Listing 20: Interface of an NFT

Rejectable Soulbound token

To obtain a RejSBT, we will get the implementation of a RejNFT that we included in Chapter 7. This implementation will be easier than a RejNFT because the RejSBT won't need to implement the $transfer$ functions. In fact, the receiver will only need to accept or reject the minting of a SBT.

```
interface ISBT {
    event Mint(address indexed _owner, uint256 indexed _tokenId);

    event Burn(address indexed _owner, uint256 indexed _tokenId);

    function balanceOf(address _owner) external view returns (uint256);

    function ownerOf(uint256 _tokenId) external view returns (address);
}
```

Listing 21: Interface of a SBT

To let receivers of an SBT reject the mint, we must add certain functions and events to the code of a regular SBT. In our proposal, we have a new mapping that will store the owner to whom we want to transfer the token: `_transferableOwners`. With this, when we mint the SBT, instead of directly transferring the ownership, we add the address of the receiver, for that SBT `TokenId`, to the `_transferableOwners` mapping.

Then, the receiver will be able to accept or reject the transfer. Besides, we have introduced another alternative: if the receiver hasn't yet accepted or rejected the transfer, the sender will still have the chance to cancel the transfer by removing the receiver from the `_transferableOwners` mapping. This new mapping stores the proposed owners of the tokens and it is defined in Listing 22.

```
// Mapping from token ID to transferable owner
mapping(uint256 => address) private _transferableOwners;
```

Listing 22: transferableOwners mapping

We only need to take into consideration the `mint()` function, where the SBT is created and doesn't have yet an owner. In fact, minting a SBT also represents a transfer of ownership, from the zero address to the receiver.

In Figure 12.2 there is a state diagram of the proposed protocol, where we can see the states of the exchange of the SBT between the sender and the receiver. Let **A** be the creator of a SBT and let **B** be the intended receiver of it. Then:

- **A** creates the token executing the operation `A.mint(B)`. With this operation, the address of B is introduced in `_transferableOwners`.
- Now **B** can accept the transfer by executing `B.acceptTransfer()` or, alternatively, it can reject the SBT with `B.rejectTransfer()`.
- Furthermore, as we have mentioned above, if **B** has not yet accepted the transfer, **A** has the ability to cancel the transfer by executing `A.cancelTransfer()`.

To enable the rejection of an SBT, it is essential not only to introduce the mapping `_transferableOwners` but also to modify the private function `_mint()`, which will add the proposed receiver of the token to the `_transferableOwners` mapping, instead of directly transfer the SBT. This private function is called from the public `mint()` function of the SBT.

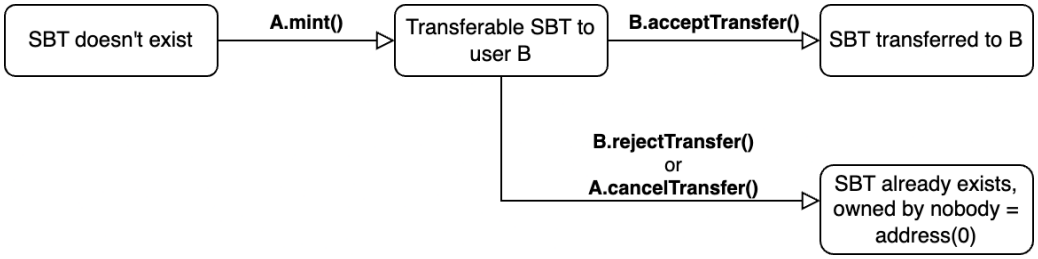


Figure 12.2: States of the RejSBT protocol

To implement the rejectable feature, we also need these new three events, that will be emitted from the corresponding functions:

- `TransferRequest`: emitted when a token is minted and proposed to be transferred.
- `RejectTransfer`: emitted when the receiver rejects the transfer of the token.
- `CancelTransfer`: emitted when the sender cancels the transfer of the token.
- `AcceptTransfer`: emitted when the transfer is really performed, changing the ownership from the sender to the receiver.

With this, the code of the private function `_mint()` is described in Listing 23

```
function _mint(address to, uint256 tokenId) internal virtual override {
    require(to != address(0), "RejectableSBT: mint to the zero address");
    require(!_exists(tokenId), "RejectableSBT: token already minted");

    _minters[tokenId] = _msgSender();
    _transferableOwners[tokenId] = to;

    emit TransferRequest(_msgSender(), to, tokenId);
}
```

Listing 23: mint() function

In addition to the three events described above, in order to complete the protocol we also need to implement the three new functions:

- `acceptTransfer()`: the receiver can call this function to accept the transfer proposal made by the sender when he mints the token. We need to check that the receiver is included in the `_transferableOwners` mapping.
- `rejectTransfer()`: the receiver can call this function to reject the transfer proposal made by the sender. Once again, this can only be done by the proposed receiver of the token.
- `cancelTransfer()`: the sender of the transfer proposal (token minter) is able to cancel it by calling this function, provided the receiver has yet to execute either the `acceptTransfer()` or `rejectTransfer()` functions.

Rejectable Soulbound token with deadline

We also have created the smart contract `RejectableSBTDeadline`, inherited from `RejectableSBT`, that lets receivers accept or reject the notification only before a deadline. This deadline represents a specific time by which the acceptance, rejection, or cancellation of the dispatch of the token must be done. To do this, when we mint a new token, we also need to provide a $deadline$ parameter. This $deadline$ will be stored for every $tokenId$ in a mapping variable. The new `_mint()` function is listed in Listing 24.

```
function _mint(
    address to,
    uint256 tokenId,
    uint256 deadline
) internal virtual {
    require(
        to != address(0),
        "RejectableSBTDeadline: mint to the zero address"
    );
    require(
        !_exists(tokenId),
        "RejectableSBTDeadline: token already minted"
    );
    require(
        deadline > block.timestamp,
        "RejectableSBTDeadline: deadline expired"
    );

    _minters[tokenId] = _msgSender();
    _transferableOwners[tokenId] = to;
    _deadlines[tokenId] = deadline;
    _states[tokenId] = State.Minted;

    emit TransferRequest(_msgSender(), to, tokenId);
}
```

Listing 24: _mint() function

We also have added an `State` enumeration that stores the possible states of a `RejectableSBTDeadline` token:

- `Minted`: the token is minted, but the receiver hasn't accepted or rejected the transfer, and the sender hasn't canceled the transfer. The deadline hasn't expired.
- `Accepted`: the receiver has accepted the transfer.
- `Rejected`: the receiver has rejected the transfer.
- `Canceled`: the sender has canceled the transfer.
- `Expired`: nobody has accepted, rejected, or canceled the transfer, and the deadline has expired.

To override the `acceptTransfer()`, `rejectTransfer()` and `cancelTransfer()` functions, we must take into consideration the `deadline` that cannot be expired, and

the token must also be in the required `State`. With this, the code of these functions is specified in Listing 25, Listing 26 and Listing 27

```
function acceptTransfer(uint256 tokenId) public virtual override {
    require(
        _transferableOwners[tokenId] == _msgSender(),
        "RejectableSBTDeadline: accept transfer caller is not the receiver of the
            token"
    );
    require(
        _deadlines[tokenId] > block.timestamp,
        "RejectableSBTDeadline: deadline expired"
    );
    require(
        _states[tokenId] == State.Minted,
        "RejectableSBTDeadline: token is not in minted state"
    );
    address from = minterOf(tokenId);
    address to = _msgSender();
    _balances[to] += 1;
    _owners[tokenId] = to;
    _states[tokenId] = State.Accepted;
    // remove the transferable owner from the mapping
    _transferableOwners[tokenId] = address(0);
    emit AcceptTransfer(from, to, tokenId);
}
```

Listing 25: acceptTransfer() function

```
function rejectTransfer(uint256 tokenId) public virtual override {
    require(
        _transferableOwners[tokenId] == _msgSender(),
        "RejectableSBTDeadline: reject transfer caller is not the receiver of the
            token"
    );
    require(
        _deadlines[tokenId] > block.timestamp,
        "RejectableSBTDeadline: deadline expired"
    );
    require(
        _states[tokenId] == State.Minted,
        "IBERejectableSBT: token is not in minted state"
    );
    address from = minterOf(tokenId);
    address to = _msgSender();
    _states[tokenId] = State.Rejected;
    _transferableOwners[tokenId] = address(0);
    emit RejectTransfer(from, to, tokenId);
}
```

Listing 26: rejectTransfer() function

```
function cancelTransfer(uint256 tokenId) public virtual override {
    require(
        minterOf(tokenId) == _msgSender(),
        "RejectableSBTDeadline: cancel transfer caller is not the minter of the
            token"
    );
    require(
        _deadlines[tokenId] > block.timestamp,
        "RejectableSBTDeadline: deadline expired"
    );
    require(
        _states[tokenId] == State.Minted,
        "IBERejectableSBT: token is not in minted state"
    );
    address from = minterOf(tokenId);
    address to = _transferableOwners[tokenId];
    require(
        to != address(0),
        "RejectableSBTDeadline: token is not transferable"
    );
    _states[tokenId] = State.Canceled;
    _transferableOwners[tokenId] = address(0);
    emit CancelTransfer(from, to, tokenId);
}
```

Listing 27: cancelTransfer() function

Rejectable SBT with a deadline that stores a confidential notification using IBE

Finally, to obtain a RejSBT with a deadline that stores a confidential notification using IBE, we will create a new smart contract, `IBERejectableSBT`, that inherits from `RejectableSBTDeadline`. This new smart contract will add three main changes: store the metadata of each notification for every `tokenId`, store the public parameters of the IBE algorithm, and add a new functionality, letting the *MW* server send the private key to decrypt the symmetric AES session key that let decrypt the notification to the receiver.

To implement the IBE algorithm, we will use CryptID.js library[6]. Using this library, first of all, the *MW* will need to set up its parameters, and its public parameters will be stored in the `IBERejectableSBT` smart contract. This setup also generates a `masterSecret`, to obtain the private keys, but this secret must be stored securely in the *MW*. Taking all of this into consideration, when we deploy the `IBERejectableSBT` smart contract, we will pass the *MW* address and the public parameters of the algorithm to the constructor of this smart contract, stored as a `bytes` type, because its size is greater than 256 bits. On the other hand, to encrypt the message, we will use AES symmetric cryptography implemented by the `crypto` native library of NodeJS [7]. This library needs a general initialization vector that will also be generated by the *MW*. These values will be stored in the smart contract (Listing 28) and can be queried publicly.

[6] https://www.npmjs.com/package/@cryptid/cryptid-js
[7] https://nodejs.org/api/crypto.html

```
// address of the middleware which will send the private key
address public middleware;

// public parameter of the AES algorithm
bytes public aesInitializationVector;
// public parameters of the IBE algorithm
bytes public fieldOrder;
bytes public subgroupOrder;
bytes public pointP_x;
bytes public pointP_y;
bytes public pointPpublic_x;
bytes public pointPpublic_y;
```

Listing 28: Middleware address and public parameters of the IBE algorithm

```
struct MessageData {
    // identity of the receiver
    address idReceiver;
    uint256 idTimestamp;
    // hash of the message in plain text
    bytes messageHash;
    // hash of the cipher of the message
    bytes encryptedMessageHash;
    // the cipher of the AES key, encrypted with the identity of the receiver
    bytes encryptedKey_cipherU_x;
    bytes encryptedKey_cipherU_y;
    string encryptedKey_cipherV;
    string encryptedKey_cipherW;
    // private key to decrypt the cipher
    bytes privateKey_x;
    bytes privateKey_y;
}

// Mapping from token ID to message data
mapping(uint256 => MessageData) public messageData;
```

Listing 29: Data stored for every notification

When a sender wants to send a confidential notification, he will encrypt the message with a symmetric AES session key, and then he will use the public parameters of the IBE algorithm to encrypt this session key using the identity of the receiver as the public key that encrypts it. This encrypted message will be sent in a secure way, off-chain, without using the blockchain, to the receiver. Then, the sender will mint a new SBT to the receiver. For every new message/SBT that we want to send, we will store the identity of the receiver, which will let the *MW* generate its private key, using the *masterSecret*. We will also store for every message the hash of the message, which will let the receiver check that its value hasn't been modified, the hash of the cipher of the message, that the sender has encrypted and sent off-chain to the receiver, and the cipher of the AES key, encrypted with the identity of the receiver. For every message, we will also need to store the private key to decrypt the AES key. This private key will be sent by the *MW* in the last step of this protocol.

```
function mint(
    address to,
    uint256 timestamp,
    uint256 deadline,
    bytes memory messageHash,
    bytes memory encryptedMessageHash,
    bytes memory encryptedKey_cipherU_x,
    bytes memory encryptedKey_cipherU_y,
    string memory encryptedKey_cipherV,
    string memory encryptedKey_cipherW
) public returns (uint256) {
  uint256 tokenId = _tokenIdCounter.current();
  _tokenIdCounter.increment();
  _mint(to, tokenId, deadline);

  messageData[tokenId] = MessageData({
    idReceiver: to,
    idTimestamp: timestamp,
    messageHash: messageHash,
    encryptedMessageHash: encryptedMessageHash,
    encryptedKey_cipherU_x: encryptedKey_cipherU_x,
    encryptedKey_cipherU_y: encryptedKey_cipherU_y,
    encryptedKey_cipherV: encryptedKey_cipherV,
    encryptedKey_cipherW: encryptedKey_cipherW,
    privateKey_x: "",
    privateKey_y: ""
  });

  return tokenId;
}
```

Listing 30: mint() function of IBERejectableSBT

Taking all of this into consideration, we will use a struct, `MessageData` to store this information for every message, and a mapping, `messageData`, that will map this data for every `tokenId`. These values are listed in Listing 29.

Once the sender has encrypted the notification, and sent it off-chain to the receiver, he will mint a new SBT calling the `mint()` function. When he calls this function, he will need to provide the address of the receiver and the timestamp (the combination of these parameters determines the identity of the IBE algorithm). He will also need to provide the deadline to accept or reject the notification, and the message hash, the cipher of the message hash, and the cipher of the AES key. The `mint()` function is listed in Listing 30.

Now, the *MW* will be listening to the smart contract `IBERejectableSBT` events, and when a receiver accepts the transfer of an SBT, that represents a message sent to him, the *MW* will generate the private key of this message and will send it to the smart contract. To generate the private key the *MW* needs the public parameters of the algorithm, the *masterSecret*, stored securely in the *MW* itself, and the identity of the receiver.

To store the private key of a concrete `tokenId`, we will use the `sendPrivateKey()` function, listed in Listing 31. In this function the smart contract will verify that the

```
function sendPrivateKey(
    uint256 tokenId,
    bytes memory privateKey_x,
    bytes memory privateKey_y
) public {
    require(
        _msgSender() == middleware,
        "IBERejectableSBT: caller is not the middleware"
    );
    require(_exists(tokenId), "IBERejectableSBT: token does not exist");
    require(
        keccak256(abi.encodePacked((messageData[tokenId].privateKey_x))) ==
            keccak256(abi.encodePacked((""))),
        "IBERejectableSBT: private key already sent"
    );
    require(
        keccak256(abi.encodePacked((messageData[tokenId].privateKey_y))) ==
            keccak256(abi.encodePacked((""))),
        "IBERejectableSBT: private key already sent"
    );
    require(
        _deadlines[tokenId] > block.timestamp,
        "IBERejectableSBT: deadline expired"
    );
    require(
        _states[tokenId] == State.Accepted,
        "IBERejectableSBT: token is not in accepted state"
    );

    messageData[tokenId].privateKey_x = privateKey_x;
    messageData[tokenId].privateKey_y = privateKey_y;

    emit PrivateKeySent(tokenId, privateKey_x, privateKey_y);
}
```

Listing 31: sendPrivateKey() function of IBERejectableSBT

caller of this function is the *MW*, and the `deadline` hasn't expired.

Once the *MW* has sent the private key, the receiver will be able to decrypt the AES key, and this AES key will let the receiver decrypt the notification (sent off-chain).

12.4 Security properties analysis

In this section, the properties related to the security, privacy, and functionality of the protocol will be evaluated. For this reason, we will evaluate Effectiveness, Fairness, Evidence transferability, Timeliness, Timestamping, Non-repudiation, and Confidentiality, the desired properties for certified notification schemes that were enumerated in section 5.1. Regarding the security properties defined in section 5.1, we have removed the *efficiency* property because it will be evaluated separately, in section 12.5, where the results and a set of experiments to evaluate the performance of the protocol are presented. We also have grouped together the properties of *fairness* and *evidence transferability* for discussion purposes.

1. **Effectiveness.** The system for certified notification presented in this chapter is effective. So, all parties will receive the expected items in case of behave according to the protocol.

   To create a new *Certified Notification*, the sender generates a new token and executes the functions following the specifications of the protocol. If all the parties execute all the steps of the exchange correctly, that is, if the receiver accepts the transfer of the token, the certified notification will be completed and the ownership of the token will prove the notification. At the end of the execution, if the receiver and the MW have followed the protocol, R will have the key to decrypt the delivered data. The state that represents this situation is *SecurityKeySent*.

2. **Fairness** and **Transferability of evidence.** The proposed protocol for *Certified Notifications* is fair. At the end of a protocol execution, the receiver has possession of the token that represents the notification and the sender can prove that the receiver has accepted the notification. Moreover, either each party has received the proper element (token and Non-Repudiation of Reception evidence) or neither party has received any useful data about the other's element, providing *strong fairness*[87]. Moreover, the evidence generated by the protocol can be verified by an external party in order to prove the outcome and the effects of the exchange. On the one hand, according to the protocol, the sender S is not going to receive the non-repudiation evidence of reception generated by the smart contract (*state* = *SecurityKeySent*) except if the receiver executes the function *AcceptTransfer()* of the smart contract and the MW executes a transaction in order to provide the IBE security key to allow the receiver to decrypt the AES key, K. On the other hand, the receiver R is able to receive the token and get access to the delivered data only if he runs a transaction in order to agree to receive the *Notification* (case *state* = *accepted*, previous to *SecurityKeySent*).

   If the parties do not follow the specifications of the protocol, that is, if they do not execute the functions *AcceptTranfer()* and/or *CancelTransfer()*, the smart contract guarantees a result that is fair for every user without the need of any intervention of a TTP, since at the deadline d the state will automatically change to *Expired*.

   We have analyzed all the cases where the protocol can lead the exchange, in order to prove the strong fairness of the protocol. We have used a state transition diagram. This diagram is formed by four final states: *SecurityKeySent*, *Canceled*, *Rejected* and *Expired* and two intermediate states: *Created* and *Accepted*. These two states cannot be final states because the protocol will eventually change the state in any case. The states and the transactions are represented in Figure 12.1.

   SecurityKeySent implies that the exchange has been completed following all the stages of the protocol, while *Canceled*, *Rejected* and *Expired* represent deliveries that have not been completed, for different reasons. The first one represents a cancellation by the sender, the second one an explicit rejection by the receiver, and the third one the rejection due to the expiration of the offer. Now, we will show how the protocol leads the delivery and the exchange of pieces of evidence to a fair situation in all cases:

- **Element C not received or invalid C received.**

  If R does not receive off-chain the element C, then he will not accept the transfer of the token. If R receives the element and the offer of the token but the $H(C)$ included in the token does not correspond with the element C received off-chain, then R will reject the notification. R can reject explicitly (*RejectTransfer(token ID)*, leading to state *Rejected* or he can do nothing and after the deadline the state will be *Expired*, invalidating the notification.

- **Element C received correctly but *AcceptTransfer()* not executed.**

  If, after the minting of the token, the receiver R does not want to receive the notification, he will not execute *AcceptTransfer* before the deadline d. Then the smart contract will set the state to *Expired*. If R decides to execute *RejectTransfer(token ID)* then the smart contract will set the state to *Rejected*. In both cases, as a consequence, R will not receive the decryption key and will not have access to the notification. For this reason, the element C received off-chain is useless. On the other hand, the sender will not have a Non-Repudiation of Reception proof generated by the receiver (reception of the token). So, the situation is fair for all the parties. The final state for this exchange will be *Expired* or *Rejected* depending on the behavior of the receiver.

- R **accepts the notification but does not receive the key** SK**.**

  After the execution of *AcceptTransfer*, the state of the notification changes to *Accepted*. If the MW behaves dishonestly and does not provide the IBE key, the state will not change to *SecuritykeySent*. Instead, after the deadline d, the state will change to *Expired*. This ensures that the NRR evidence will not be generated, as the token will not be associated with R's wallet.

- R **accepts the notification but the** MW **provides an invalid Security Key,** SK**.**

  After the execution of *AcceptTransfer* the state of the notification is *Accepted*. If the MW tries to act dishonestly by providing an invalid IBE key, (we will call it SK') R will be able to prove this fraud attempt. When the MW provides the false SK' then R will use it to obtain an invalid K, K', since $K' = IBE.decrypt_{SK'}(k)$. R will also use the element C received off-chain together with this key K' to obtain the message. When the false K, K' is used, the element obtained will be a false m, called m', being $m' = AES.decrypt_{K'}(C)$. Then, the receiver R will check if the hash of m' corresponds to the element $H(m)$ included in the token. When the comparison does not stand, R knows that the MW has not acted honestly. To prove this misbehavior, R must show C and m'. Since $H(C)$ is included in the token, everybody can verify that it is a valid element. Then it can be proved that the SK' provided by the MW does not correspond to the key that allows to decrypting of the message. With this verification, fairness is ensured. Moreover, since the misbehavior of the MW could be detected, and taking into account that once the proof of the fraud is revealed the reputation of the MW is destroyed, it is assumed that the MW will not try to act dishonestly.

- **The receiver accepts, receives the corresponding key, and decrypts the notification.**

  If the receiver executes *AcceptTransfer* to accept the notification, then the MW will provide the IBE security key SK. With this key K can be revealed, since $K = IBE.decrypt_{SK}(k)$. Then, R will use the element C received off-chain together with this key K to obtain m, $m = AES.decrypt_K(C)$. The receiver R will check if the hash of m corresponds to the element $H(m)$ included in the token. After this verification, R has access to the notification. The final state of the protocol is *SecurityKeySent* and R is the owner of the token. The transfer of the token represents the NRO and NRR pieces of evidence.

- **S cancels the transfer.**

  After the execution of *Protocol 1: Notification Release*, before the deadline and before the acceptance by R, if S wants to cancel the notification, he can do it by executing the *Cancel* function. This function can be useful in case of a mistake by S. The notification can be canceled without any consequences to the receiver R. Since R has not performed any action, there will be no damage to him. In this case, the notification is not completed and neither NRO nor NRR pieces of evidence are generated.

According to the previous evaluation, the fairness provided by the protocol is *strong fairness*. Although the execution of the protocol can lead to different states, the protocol does not allow any circumstance where a user could get contradictory evidence since the state is only updated by the smart contract. Therefore, it is not possible to do an action that could lead to an unfair situation.

The proofs generated during the execution of a protocol run can serve as evidence and can be submitted to an external arbiter. It is essential to note that users cannot get contradictory proofs, and valid evidence is only created by following the execution sequence of the smart contract. The evidence can be evaluated by an external arbiter who can determine whether the notification has concluded successfully or not. In addition to that, its transferability is easy, because the proofs generated during the exchange are all stored on the blockchain. Therefore, since the blockchain is immutable, it is not possible to change the content of any message and, thus, the scheme provides message integrity. It is also possible to deduce the exact point in time when the delivery took place from the timestamp of the block where the transaction was included.

3. **Temporal parameters: Timeliness and Timestamping.** A successful notification will always be completed before the deadline d.

   If the notification is not successful we have different situations depending on how the exchange has been performed. If the receiver R does not accept the notification and executes *RejectTransfer*, then the delivery will immediately lead to *Rejected* state. If the receiver R does not accept and does not make any action, the delivery will lead to the state *Expired* at the deadline d. The same happens if, after the acceptance of the notification by the receiver, the MW does not provide the security key, SK.

Moreover, we have to take into account that the blockchain timestamps all transactions performed on it.

4. **Non-repudiation.**

   The notification protocol must provide a Non-Repudiation of Reception evidence and a Non-Repudiation of Origin evidence too.

   - Regarding the Non-Repudiation of Origin (NRO) evidence, the sender S cannot deny having executed the functions included in *Protocol 1* to create the *notification* since there is a *mint* of a token by his address containing ID, d, $H(m)$, $H(C)$ and k. The related smart contract can prove that the steps of *Protocol 2* have been executed and that the final state of this notification is *SecurityKeySent*, proving that R has received the key to decrypt the message. The pieces of evidence are related to a notification whose hash is included in the token, $H(m)$.
   - Concerning the Non-Repudiation of Reception (NRR) evidence, R is not able to refuse the notification acceptance, because an *AcceptTransfer* transaction from his address is stored on the blockchain. This transaction accepts the reception of the Notification and, according to that, the smart contract changed the *state* of the notification to *SecurityKeySent* after the delivery of SK by the MW.

5. **Confidentiality**. A notification will be confidential if only the receiver that accepts the notification can access the delivered message. For this reason, the data cannot be included in clear text in any transaction and must not be registered on the blockchain. The data cannot be a parameter in the smart contract functions and the smart contract cannot gain access to the decryption key if it has access to the encrypted message.

   In *Protocol1* the sender encrypts the message $C = AES.encrypt_K(m)$ and then encrypts the key K using an IBE scheme $k = IBE.encryption_{ID}(K)$. Neither the notification message m nor the encrypted message C are included in a blockchain transaction. When *Protocol 2* is executed, the smart contract will gain access to k, and then to K, but since the smart contract does not have access to C, he will not be able to decrypt the notification message. Thus, the protocol achieves the confidentiality property.

6. **Race Conditions and Trust in Layer 2 Solutions.**

   While the proposed protocol effectively handles security properties like fairness, non-repudiation, and confidentiality, it is essential to consider potential race conditions in nodes that include transactions in the blockchain, especially when using a Layer 2 (L2) solution like Polygon PoS.

   In L2 networks, a limited number of validator nodes may introduce trust dependencies that differ from the decentralized nature of Ethereum's mainnet. These validators are responsible for confirming and including transactions in the blockchain. If validators act dishonestly or if there are synchronization issues between them, it may lead to race conditions or delayed transactions, potentially impacting the protocol's integrity.

Therefore, while L2 solutions provide cost efficiency and scalability, they also introduce a level of trust in the validator set. The middleware in our protocol, while not a traditional TTP, must be considered in this context. Its behavior and reliability are crucial, as it ensures that the security keys are correctly issued to the receivers.

To mitigate these risks, it is vital to implement robust validator selection mechanisms and ensure high standards of validator performance and reliability in the chosen L2 network. Additionally, the protocol's ability to trace and verify actions on the blockchain helps maintain transparency and trust, even in a more centralized L2 environment.

12.5 Performance analysis

The implementation of this protocol was tested for performance using the hardhat-gas-reporter plugin[8], an integral part of the Hardhat development environment. This plugin facilitates the calculation of Gas usage per unit test, using metrics for method calls and deployments.

To validate the correctness of the protocol, we designed tests that we executed using the aforementioned hardhat-gas-reporter plugin, which provides us with metrics for the evaluation of the efficiency of the system in terms of gas cost.

In the Ethereum Virtual Machine (EVM) compatible blockchains, gas execution costs play a crucial role for the efficiency of the execution of smart contracts, and subsequently, they are a significant concern in the development of this service.

To calculate the gas cost in US Dollars, we utilized the Polygon PoS network, a layer 2 scaling solution for Ethereum that aims to provide faster and cheaper transactions. Polygon PoS generally incurs lower gas costs compared to Ethereum due to its distinct fee structure. Furthermore, as a layer 2 scaling solution, it batches transactions and settles them on the Ethereum mainnet in a single transaction, reducing the overall gas costs. This makes Polygon PoS a good alternative to Ethereum, especially to for users requiring frequent or high-value transactions.

Based on the test results, we analyzed the gas cost of all the functions employed in the protocol. Figure 12.3 represents these costs, according to the Hardhat–gas–reporter plugin. Notably, these costs are denominated in gas units. To determine the exact transaction price, we require information about the current gas price in MATIC (the native currency of the Polygon PoS) and the MATIC to US Dollars exchange rate. For this analysis, we used the MATIC price from November 14, 2023, which is 0.90 USD/MATIC. This conversion to USD is provided for reference purposes only and reflects the rates at the time of this study. For comparison with other results, it is recommended to use the gas units.

Analyzing the gas cost of the smart contract deployment and the execution of the function, we notice that the most expensive function, in terms of gas cost, is the deployment of the `IBERejectableSBT` smart contract itself, which costs 0.30 US Dollars in the execution of our tests and maintaining an average cost of 0.48 US Dollars over the past year. In contrast, the remaining functions (`mint()`, `acceptTransfer()`,

[8]https://www.npmjs.com/package/hardhat-gas-reporter

Methods		128 gwei/gas	0.90 usd/matic
Contract	**Method**	**Avg**	**usd (avg)**
IBERejectableSBT	acceptTransfer	83762	0.01
IBERejectableSBT	cancelTransfer	50954	0.01
IBERejectableSBT	mint	540612	0.06
IBERejectableSBT	rejectTransfer	50710	0.01
IBERejectableSBT	sendPrivateKey	172718	0.02
Deployments			
IBERejectableSBT		2604207	0.30

Figure 12.3: Gas cost of the two-steps certified notifications smart contract

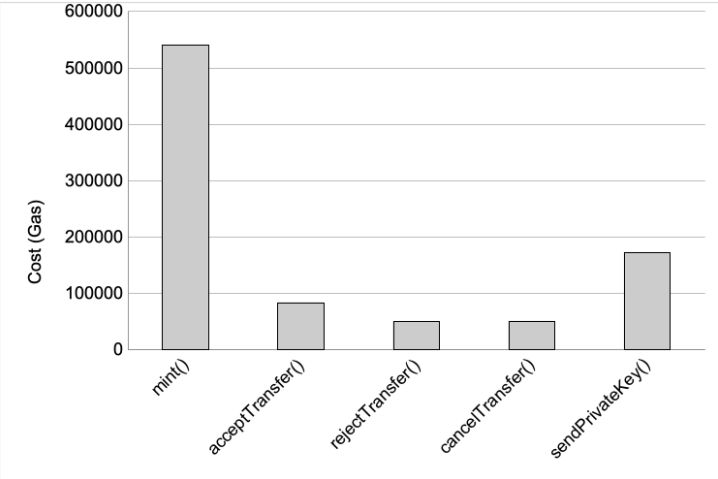


Figure 12.4: Gas cost of the RejSBT smart contract

`rejectTransfer()`, `cancelTransfer()` and `sendPrivateKey()`) are relatively inexpensive, with average costs of 0.1, 0.015, 0.009, 0.009 and 0.03 US Dollars, respectively (see Table 12.2).

Considering these results, alongside a notable peak in gas costs observed in early November 2022, the costs associated with the defined and implemented protocol are reasonably affordable, meeting the desired properties for handling confidential certified notifications.

12.6 Conclusions

The protocol presented in this chapter uses blockchain and tokens to represent the delivery of data. Due to the nature of these technologies, we obtain some essential

Table 12.2: Average costs of the two-steps certified notifications smart contract

	US Dollars
IBERejectableSBT deployment	0.47699
acceptTransfer	0.01574
cancelTransfer	0.00958
mint	0.10164
rejectTransfer	0.00953
sendPrivateKey	0.03247

security properties. However, the standards related to tokens defined up to the current date do not fulfill the requirements for certified notifications. For this reason, we have enhanced our previous proposal for tokens, RejNFTs, that provide the receiver the ability to reject a token delivery proposal and then we have proposed a new kind of token. Taking into account that the notifications cannot be transfered by the receiver we have presented a token proposal, RejSBTs. The transference of the token represents the delivery process and the token itself can be used to obtain Non-Repudiation of Origin and Non-Repudiation of Reception evidence.

The receiver of a notification must accept or reject it before accessing its contents. Using the proposed tokens together with IBE Cryptography, we have designed a confidential certified notifications protocol that allows the delivery with only one action performed by the sender and one by the receiver, obtaining a very efficient proposal. The IBE scheme is used to link the token with the ID of the notification and helps to keep it confidential.

Finding efficient and secure fair exchange protocols has been an open challenge [178]. The solution presented in this chapter is a significant contribution that can securely solve the problem with a 2-step protocol. In this way, the sender just has to perform one step to submit the message and, then, the non-repudiation evidence will be available on the blockchain. In contrast to the regular three-step solutions where the sender also has to intervene in the third step to open the message to the receiver. Taking this into account, as further works, we can study how our certified notification protocol can be integrated as a new fair messaging exchange paradigm as an external system in other software applications [179].

The security of the protocol has been evaluated proving that the protocol fulfills the desired properties including effectiveness, fairness, transferability of pieces of evidence, timestamping, non-repudiation, and even confidentiality of the delivered message. Moreover, the performance of the system has been also evaluated in order to check the viability of the proposal, obtaining interesting results.

Despite the improvement done in this protocol, reducing the number of steps to two, where each actor is only involved in one step, there are also some limitations. For example, the use of a Middleware introduces a third party that can represent a single point of failure. Despite the fact that this middleware must participate fairly in the exchange, and its malfunction is controlled by the protocol itself, if it stops working, a lack of service would occur. In any case, the protocol has the capacity the verify the Middleware behavior in such a way that any action made by the Middleware that is not

in accordance with the protocol specification can be detected and proved because it can be traced in the blockchain records.

The RejSBTs proposed in this chapter are not only useful for their usage in certified notifications, but they have great potential and in the future, they will be useful in the development of other applications with similar requirements to those of the certified notifications. These tokens could find application in secure access control systems, identity verification processes, financial transactions, and even healthcare data management, ensuring a versatile and reliable solution across a spectrum of fields demanding heightened security measures.

CHAPTER 13

TWO-PARTY CONTRACT SIGNING PROTOCOL

In this chapter, we explore a contract signing protocol as a trusted service, which fundamentally relies on the fair exchange of signatures from each participating party. Traditionally, such services have depended heavily on Trusted Third Partys (TTPs). However, these entities often introduce complications that can impede the broader adoption of these services.

After a thorough analysis of several certified notification protocols in previous chapters, we present a new protocol that utilizes blockchain-based technologies. This approach effectively eliminates the need for TTPs while still fulfilling the essential requirements for secure and reliable contract signing. Our protocol supports the execution of both public (non-confidential) and private (confidential) contracts, providing a versatile solution adaptable to various legal and business needs.

Moreover, the design of our protocol considers future enhancements, specifically its extension to include multiparty contracts. This expansion, which will be addressed in the next chapter, will allow for more complex contractual agreements involving multiple stakeholders, thereby broadening the applicability of our blockchain-based solution. By removing TTPs and leveraging the inherent security properties of blockchain, our protocol promises to streamline the contract signing process, making it more accessible and efficient for all parties involved.

13.1 Contribution

Electronic contract signing, described in section 5.2, is a trusted service offered to users who want to obtain a signed copy of a contract from another user. Protocols for contract signing have to provide irrefutable evidence to the parts to prove, at the end of the exchange, whether the contract is signed and, if it is the case, the terms of the contract. The signature of a contract is handled as a fair exchange of values: the proposer has an item (the text of a contract together with a non-repudiation or commitment token) to be exchanged for a recipient's item (a non-repudiation token bounded to the text of

the contract). The exchange of these items will be fair if, at the end of its execution, all users have received the item they expected to receive or no user has received it [114].

However, in practice, the implementation and acceptance of this type of entity is an obstacle to extending the use of protocols in the network. On the one hand, it is difficult to have TTPs that are really reliable for any user in the network and that have a defined framework of action (e.g. the electronic documents generated by the TTP have to be accepted to resolve disputes in a court of law in different countries). Moreover, TTPs can also cause problems at a technical level (e.g., they can cause bottlenecks from a communications point of view), lack of efficiency in the protocols (e.g., slow down the resolution of problems) and increase the cost of the execution of the protocol (e.g. charging high rates for the provision of services). Besides, they are a very sensitive point in the network since they play an important role in the security of electronic protocols and their reliability is a problem that needs attention because the security of the exchange can be broken if the TTP has any vulnerability. Although TTPs are still a basic actor in fair exchange protocols, currently, with the advent of blockchain technologies and smart contracts, TTPs could be replaced or complemented by this new know-how, which opens a range of new possibilities to find effective solutions to the electronic versions of the protocols that fulfill the generic pattern of fair exchange of values.

This protocol aims to show how blockchain technology and smart contracts can introduce a new paradigm to deal with contract signing operations. By using this technology, we can reduce or even remove the role of the TTPs inside such protocols. This means that this new technology allows us to define transactions with predetermined rules (written in a contract) in a programmable logic that can guarantee a fair exchange between parties with an initial mutual distrust. This feature prevents parties from cheating each other and discharges the need for intermediaries with the consequent reduction of delays and commissions for their services. The revealing power of the blockchain is further enhanced by the fact that blockchains naturally incorporate a discrete notion of time, a clock that increases each time a new block is added. The existence of a trusted clock/register is crucial to achieve the property of fairness in the protocols.

In this chapter, we present a contract signing protocol that avoids the involvement of Trusted Third Partys while satisfying the security and privacy requirements, including the confidentiality of the contract.

13.2 Properties and requirements

Some ideal properties for optimistic fair exchange are effectiveness, fairness, timeliness, non-repudiation and verifiability of the third party. Two additional properties are efficiency and privacy. These properties are described in section 5.1. During the design process we have determined some other requirements for this particular contract signing protocol:

- The process requires three steps, according to [112].
- The designed protocol uses a similar approach to the solution we presented for certified notifications in Chapter 8, but they must differ in the way to get

access to the provided information. In a certified notification the receiver cannot access the data until the finalization of the exchange and in the case of contract signing all signers have to be able to read the contract before its signature for both non-confidential and confidential contracts.

- A contract, in order to be signed, requires the signatures of all the involved signers. When a signer does not accept the contract, then the exchange has to be canceled. In multiparty exchanges, partially signed contracts are not allowed. This is another difference with the protocol presented in Chapter 8 because partial notifications are allowed.
- If the contract requires more than two signers, the smart contract has to ensure that all the signers read the same contract, even if the smart contract cannot access the contents of the contract.
- Even if the contract is confidential, any signer has to be able to prove to an external party, using the evidence stored in the blockchain, that the contract is signed.

13.3 Protocol design

The proposed system presents solutions for both confidential and non-confidential (or public) contract signing between two signers. The proposals consider the following actors with these roles:

- **Proposer or Signer** A. The user that generates the contents of the contract, chooses the signer B and starts the exchange. A has to generate the contract and send it to B in order to obtain B's signature on the contract and finish the exchange.
- **Receiver or Signer** B. The user that receives the contract to sign. The receiver must accept the contents of the contract and provide a signature on it.
- **Smart Contract**. Contract deployed on the blockchain that can manage the exchange and ensure fairness.

All the participating actors have blockchain addresses and are able to communicate with the smart contract. These entities interact as follows: first A chooses whether he wants to propose the signature of a non-confidential or a confidential contract. Then, in both cases, the parties execute a three-step exchange to provide the contract, to sign it and also to generate the non-repudiation elements. Both proposed protocols follow a three-step on-chain exchange using the smart contract. The confidential proposal uses additional steps to guarantee that the smart contract will not have access to the contents of the contract. The proposed protocols will be explained in subsection 13.3.1 and subsection 13.3.2. Table 13.1 includes the notation used in the definition of the proposed protocols.

We use states to manage the exchange process. The state diagram for the confidential contract signing protocol is depicted in Figure 13.1. We have programmed the smart contracts and checked it on the Ethereum blockchain.

Table 13.1: Notation of the two-party contract signing protocol

Notation	
C	Contract
X, Y	Concatenation of messages X and Y.
$p \blacktriangleright e.f$	Execution of function f of e by p.
$term$	Timeout for A to finish the exchange.
D	Deposit sent to the Smart Contract (ethers).
x_A	A's contracting private key.
y_A	A's contracting public key.
x_B	B's contracting private key.
y_B	B's contracting public key.
g, p, g	System parameters.
r	Encryption secret element used by A.
s	Encryption element used by B.
$M_1 = g^r_{mod\,p}$	First part of ElGamal encryption of the contract.
$M_2[i] = c[i]$ XOR key	Second part of ElGamal encryption of the contract.
$key =$ $random.seed(hash(r))$	Encryption key
$c[i]$	Fragment of the contract.
$h()$	Hash Function
$challenge$	Challenge sent by B to A
w	Response to the challenge.

13.3.1 Non-confidential two-party contract signing protocol

In this first proposal, we consider that confidentiality is not required or even desired. The proposer A executes the first step of the protocol using the DApp to register the content of the contract on the blockchain. At this point, the receiver can have access to the contract, and the transaction remains stored in the blockchain. The proposer will make a new transaction representing the signature on the contract in a third step, provided that the receiver B would have made a previous transaction signing the content of the contract.

1. Subprotocol 11. Creation of a Contract Signing Operation. The proposer (signer A) and creator of the contract, uses the smart contract to publish in the blockchain the contents of the contract. Other parameters of this transaction are the address of the receiver and the deadline for the signature of the contract to be completed. Moreover, a deposit can be required in this step. The amount will be included in the transaction.

Subprotocol 11: Create a Contract Signing Operation

1. $A \blacktriangleright SM : c, B, term, D$
2. SM: $State = Created$

2. Subprotocol 12. The proposed signer B, in case of accepting to sign the contract, makes a transaction expressing his will. B can individually decide whether to

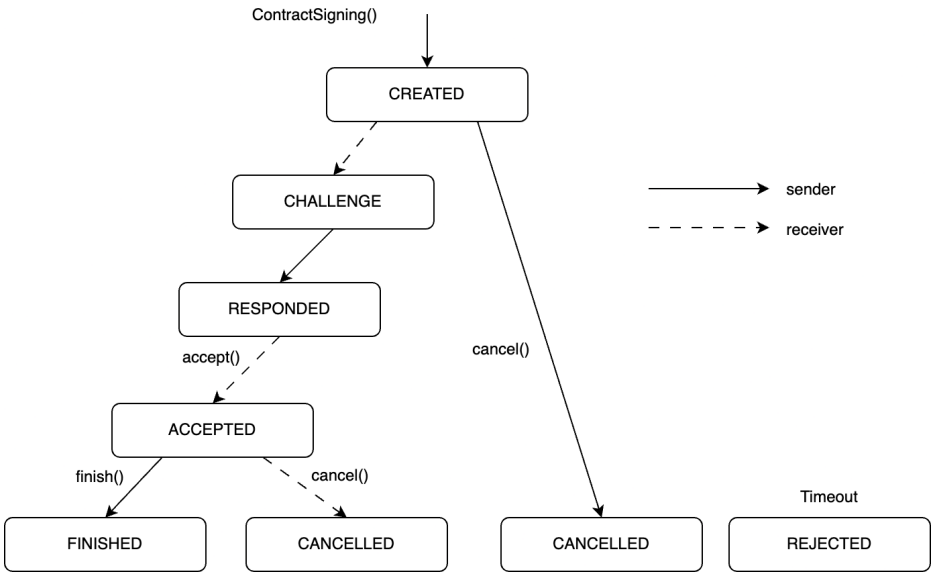


Figure 13.1: States of the two-party contract signing protocol

accept to sign the contract or not, by executing the corresponding function of the smart contract before de deadline *term*. In case the signer B does not accept the contract, the exchange will be Rejected. The smart contract checks the identity of the user who has invoked the function. In the prevision of multiparty exchanges, if a signer does not accept before *term*, the signature of the contract would be canceled for all signers.

Subprotocol 12: Accept the Contract

1. $B \blacktriangleright SM.accept$
2. SM:
   IF$(now < term, Id = B$ AND $State == Created)$
   $State = Accepted$

3. Subprotocol 13. Finally, before the expiration of the deadline, if the proposed signer B has accepted/signed the contract, the proposer A can finish the signature process, executing the *finish* procedure to sign the contract. As a consequence, the smart contract publishes the signature. If the execution of the exchange requires a deposit, the smart contract returns the amount to the sender.

After the deadline, if the three steps have not been executed properly, the state of the exchange is not *finished* and then both parties can access a function in the smart contract to request the cancellation of the transferred elements.

1. Subprotocol 14. Cancellation of acceptance. This function is requested by signer B, if the proposer (signer A) does not publish its signature when the receiver has accepted the contract.

Subprotocol 13: Finish the Contract Signing Process

1. $A \blacktriangleright$ SM.finish(M)
2. SM:
   IF($now < term$, $Id = A$ AND $State == Accepted$)
   $State = Finished$
   Contract C is stored in the blockchain
   Deposit D is refunded to A

Subprotocol 14: Cancellation of Acceptance

1. $B \blacktriangleright$ SM.cancel()
2. SM:
   IF($now >= term$, $Id = B$ AND $State == Accepted$)
   $State = Canceled$

2. Subprotocol 15. Cancellation of proposal. This function is requested by the proposer if the receiver has not accepted the contract.

Subprotocol 15: Cancellation of Proposal

1. $A \blacktriangleright$ SM.cancel()
2. SM:
   IF($now >= term$, $Id = A$ AND $State == Created$)
   $State = Canceled$

In both cases, the smart contract checks the identity of the user and the deadline. The smart contract generates a transaction to point out that the signature of the contract has been canceled. In the first case, the proposer will not receive the refund of the deposit (this way, the deposit is useful to motivate the proposer to finish the exchange before the deadline).

After the exchange protocol, the text of the contract and the addresses of the signers are public and are stored in the blockchain. If after the deadline *term*, the proposer A has not proceeded with the finalization/signature of the contract, there is evidence of the cancellation of the contract.

Since the contract is included in a transaction, it will be registered on the blockchain. Thus, the contract is not confidential. This protocol is executed entirely over Ethereum, so no off-chain communication between the parties is required. This way, there is no need for extra communication channels between the parties.

13.3.2 Confidential two-party contract signing protocol

In this second proposal, confidentiality is required. This solution provides fairness to the exchange of signatures even when the smart contract does not know the content of the contract and it is not registered on the blockchain. The proposer A executes the first step of the protocol using the DApp to register the encrypted content of the contract on

the blockchain. This encryption has to guarantee that only the signers can access the content of the contract. Moreover, in the prevision of multiparty contracts, this step has to be designed in a way that the smart contract executes a verification to ensure that the contract that each signer can decrypt is the same contract. To execute this step all signers have to generate a pair of keys, that will be called Contracting Keys.

At this point, the receiver does not have access to the contract, so the smart contract includes new steps to allow the receiver to access the content of the contract before its acceptance. These steps take the form of a challenge-response stage. Thanks to this response, the smart contract verifies that all signers can access the contract, even though the smart contract is to able to know its content. Then, the signer B has to accept the contract by means of a transaction. This transaction remains stored in the blockchain. Finally, the proposer A will make a new transaction representing the signature on the contract in a third step, provided that the receiver B would have made a previous transaction signing the content of the contract.

The cryptographic algorithms used in the design of the protocol are:

- ElGamal Encryption. The encryption and decryption processes are performed off-chain.
- Schnorr Zero-Knowledge Proofs (ZKPs). The verification of the ZKP is performed on-chain by the smart contract. The description of the Solidity implementation of the ZKP is not included in this chapter.

The proposer of the contract A and the proposed signer B will exchange the confidential contract and the evidence of non-repudiation following the steps of an exchange protocol.

1. **Creation** Subprotocol 16. Proposer A encrypts the contract to be signed, C using ElGamal encryption with a secret encryption element r and its private key from a pair of contracting keys created specifically for this exchange, x_A, y_A (if it is necessary the contract can be fragmented). To do this, A creates a smart contract by invoking the factory constructor function provided by the service provider, including as parameters the encrypted message $\{M_1, M_2\}$, the public contracting key of the proposer y_A, the address of the proposed signer, B and the deadline $term$. This deadline specifies the valid period for the proposer to finish the exchange and the moment from which the receivers will be able to have evidence of the signature of the contract or obtain evidence of the cancellation of the process that has not been completed by the proposer. Optionally, a deposit D or payment for the service can be included in this stage.

2. **Challenge** Subprotocol 17. Before *term*, the signer B has to accept the contract and sign it. But first, the proposer A has to allow access to the message to B while keeping it confidential for the rest of the world. That is why the plain contract cannot be included in a transaction nor stored in the blockchain. A must generate the necessary elements so that he can confidentially deliver the key to decipher the contract, (that is, the proposer sends the key that has been committed to send at the beginning of the exchange. See Subprotocol 18) and also that the smart contract can ensure that the signer B can access the contract. Signer B generates

Subprotocol 16: Step 1. Creation

1. A : generation of $C, r, y_A = g^{xA}_{modp}$
2. A : $key = random.seed(hash(r))$
3. A: Fragmentation of C, if necessary.
   for $i = 1$ to c.length
   $M_1 = g^r_{modp}$
   $M_2[i] = C[i]$ XOR key
   $key = random.seed(hash(key))$
4. $A \blacktriangleright SM : SM.creation(M_1, M_2, B, term, y_A, g, p, D)$
5. $SM : State = Created$

its own pair of contracting keys x_B, y_B and sends its public contracting key y_B, a variable that will be used as a challenge, and its private contracting key encrypted with the public contracting key of the proposer A $\{Z_1, Z_2\}$.

Subprotocol 17: Step 2. Challenge

1. B : generation of $y_B = g^{xB}_{modp}$, s
2. B : Z_1=g^s_{modp}, Z_2=$x_B * y^s_{Amodp}$
3. $B \blacktriangleright SM.challenge(Z_1, Z_2, y_B, challenge)$
4. SM :
   IF$(now < term, Id = B$ AND $State == Created)$
   $State = Challenge$

3. **Response** Subprotocol 18. Then, the sender generates for this signer a response to the challenge in the form of a ZKP using the secret element r used in the encryption of the message in the first step of the protocol, the challenge and B's private contracting key. The Smart Contract will store the parameters. With this response, the Smart Contract can verify, by means of the stored data, that Signer B will be able to know the secret element that will allow him to decipher the message. However, the Smart Contract will not know this element and therefore the message will remain confidential.

4. **Accept** Subprotocol 19. The proposed signer B decides whether to accept the contract or not. If he accepts to sign the contract, publishes a message expressing his will. If the signer B does not accept before *term*, a rejection is assumed ($State = Rejected$), otherwise, the contract has been accepted by the signer B ($State = Accepted$). In the case of multiple signers, each possible signer can decide individually whether to accept to sign the contract or not, by executing the corresponding function of the smart contract before de deadline *term*. In case, the signer B does not accept the contract, the exchange will be canceled. In the prevision of multiparty exchanges, If there is a signer that does not accept before *term*, the signing of the contract will be canceled for all signers.

Subprotocol 18: Step 3. Response

1. A : generation of w=t'$r + challenge * x_{B\,mod q}$
2. $A \blacktriangleright SM : response(w)$
3. SM :
   IF$(now <= term)$, $Id = A$, $State == Challenge$ AND
   $(g^w == g^r * y_B^{challenge}{}_{mod p})$ $State = Responded$
4. B :
   $key = random.seed(hash(r))$
   for $i = 1$ to n
   $C[i] = M_2[i]$ XOR key
   $key = random.seed(hash(key))$

Subprotocol 19: Step 4. Accept

1. $B \blacktriangleright$ SM.accept
2. SM:
   IF$(now < term)$, $Id = B$ AND $(State = Responded)$
   $State = Accepted$

5. **Finish** Subprotocol 20. Finally, before the expiration of the deadline, if the proposed signer B has accepted/signed the contract, the proposer A can finish the signature process. The proposer can execute the *finish* procedure to sign the contract. As a consequence, the smart contract concludes the signature process. If the execution of the exchange requires a deposit, the smart contract returns the amount to the sender.

Subprotocol 20: Step 5. Finish

1. $A \blacktriangleright$ SM.finish(M)
2. SM:
   IF$(now < term)$ AND $(State = Accepted)$
   $State = Finished$
   Deposit D is refunded to A

After the exchange protocol, the contract C is confidential and is not stored on the blockchain. However, the Smart Contract can prove that signers have the element to decipher the message. Moreover, the encrypted contract and the addresses of the signers are public and are stored on the blockchain.

If after the deadline *term*, the proposer A has not proceeded with the finalization/signature of the contract, the other signer may have evidence of the cancellation of the contract. After the deadline, if the steps have not been executed properly, the state of the exchange is not *Finished* and then both parties can access a function in the smart contract to request the cancellation of the transferred elements. Protocols described in the previous section Subprotocol 14 and Subprotocol 15 could be executed.

13.4 Conclusions

Previous solutions for fair contract signing are mainly based on the intervention of a TTP that acts as an intermediary between the signers. In this model of fair exchange, both parties obtain the expected item from the other or neither obtains what was expected. That is, either all the signers have received the acceptance of the contract from the other signers or neither party obtains the desired item. In these traditional solutions, the TTP can intervene to guarantee the fairness of the exchange if some participant misbehaves.

This protocol presents two alternatives for contract signing operations using blockchain-based fairness.

On the one hand, the first solution allows users to sign non-confidential contracts. It supports the exchange of signatures between signers and guarantees the fairness of the exchange without requiring the intervention of any TTP to guarantee the security properties since the actions of the different actors are recorded in the blockchain and if any actor does not fulfill the protocol, the smart contract will generate the corresponding evidence to preserve fairness.

On the other hand, the second solution is a fair exchange protocol that allows users to sign contracts that require confidentiality. To achieve this property, and taking in mind the future extension of the protocol to multiparty contract signing, we have included in the protocol a challenge-response stage and a Zero-Knowledge Proof.

Both protocols preserve the properties of limited Timeliness (involved parties can be certain that the protocol will be completed at a certain finite point in time), Transferability of proofs, and Non-repudiation.

CHAPTER 14

CONFIDENTIAL MULTIPARTY CONTRACT SIGNING PROTOCOL

After introducing a confidential and a non-confidential two-party contract signing protocol in Chapter 13, this chapter expands the discussion to multiparty scenarios. Managing signed information is particularly challenging when it involves the atomic exchange of digitally signed documents. Multiparty contract signing, an application of fair value exchange, needs the exchange of signed versions of a contract. As mentioned before in this thesis, this trust service has depended on the involvement of a Trusted Third Party (TTP), which has been a major limitation in the general adoption of electronic contract signing protocols. Additionally, the complexity of multiparty protocols significantly surpasses that of two-party agreements, with a recurring demand from signers for protocols that ensure confidentiality.

Responding to these challenges, the protocol described in this chapter leverages blockchain-based technologies and smart contracts to forge an electronic contract signing protocol that requires no TTP at any procedural stage. This protocol fulfills the essential requirements for effective contract signing services and allows for the signing of confidential contracts. Moreover, this solution offers substantially greater efficiency compared to previous blockchain-based fair exchange protocols, thanks to the utilization of Elliptic Curve Cryptography (ECC).

14.1 Contribution

Multiparty contract signing is a very common e-commerce operation. In multiparty contract signing several users, called signers, generate and sign a contract. One of them will perform as a proposing signer presenting the contract to be signed and the list of signers. Then the signers have to follow a protocol to add their signatures to the contract and send the signed contract to the remaining signers. The protocol must ensure that the signers cannot have access to partially signed contracts. That is, all signatures on the contract have to be canceled in case a signer does not sign the contract.

In practice, this objective is difficult to achieve without the use of a TTP. If a problem arises during the exchange or one of the users does not follow the phases of the protocol it is possible that some signers have received the contract signed by other users while other signers have not received it. The TTP can then solve this situation.

When the TTP is removed, the blockchain can be used to manage the exchange. Blockchain is a distributed registry of data, with the property of immutability of the stored data that can be verified through a decentralized structure. Blockchain will not only replace the role of the TTP but also it does it with better results: it eliminates the need to agree on which TTP to use, it reduces execution costs, it avoids queues and delays waiting for the TTP to solve the exchange and solves the issues related with transparency of the intermediary.

Using blockchain the costs of having a TTP are replaced by the cost of the use of the blockchain. Using smart contracts the execution costs are related to the computational cost of the operations performed by the smart contract. To achieve the goals of the protocol we have used complex algorithms. However, we have kept the cost reduced thanks to the use of Elliptic Curve Cryptography.

This chapter proposes a multiparty contract signing protocol that does not require the participation of TTPs but satisfies the requirements for fair exchange and contract signing related to security and privacy, allowing the signature of confidential contracts. We have used Elliptic Curve Cryptography in the design of the protocol to obtain the desired security properties with reduced execution costs.

14.2 Properties and requirements

In this section, we include the list of desired properties for fair exchange protocols. These properties are included in several papers, as the one explained in section 5.1. The properties are fairness, effectiveness, non-repudiation and timeliness. Since we are not using TTPs we are not considering the properties related to the TTP as Transparency or Verifiability.

- Fairness: in a fair protocol a party can't obtain the desired element while another party does not obtain it.
- Effectiveness: a protocol achieves effectiveness if when all the parties follow the phases of the protocol then the contract is signed.
- Non-repudiation: no party can deny its involvement in the steps of the protocol.
- Timeliness: no party has to wait indefinitely to know the final state of the exchange.

We have evaluated these usual requirements for fair exchange protocols during the design process we have determined some other requirements. Additionally, we have taken into account two more properties, confidentiality and efficiency, and the following requirements.

- Contract signing is based on a three-step exchange, according to [112], to *propose*, *accept* and *confirm* the signature of the contract.

- A contract signing process has some similarities with a certified notification process, as the protocols proposed in Chapter 8, Chapter 9 and Chapter 12, but they have important differences in the management of the transferred information. While for notifications the recipient will not be allowed to read the transferred data until the end of the exchange in the signature of a contract all the users must have access to the contents of the contract from the beginning, in order to decide if they want to sign the contract or not.
- In a multiparty protocol with more than two signers, all the signers must sign the same contract. A function of the smart contract is to ensure, even when it does not have access to the contract, that all the signers have accepted and signed the same contract.
- A valid signed contract must include the signatures of all the signers. If any signer does not accept the signature of the contract, then the signature process must be canceled. That is, a contract has to be signed by all the specified signers while partially signed contracts are not valid. This has to be taken into account when the protocol is designed, being a main difference with certified notification protocols, like the protocols explained in previous chapters, where partial notifications are allowed.
- Regarding the transferability of pieces of evidence, all the signers must be able to prove to any external arbiter, using as evidence the data stored in the blockchain, that the contract has been signed. This evidence has to be provided even when the contract signing protocol is confidential.

14.3 Protocol design

The scheme that we propose in this chapter allows the fair signature of confidential contracts. Even if the system does not involve a TTP, it fulfills the properties and additional requirements listed in section section 14.2. The actors of the protocol are:

- Proposing Signer $Alice$. Generator of the contract. The proposing signer also sets the group of signers R. This is the user that starts the execution of the protocol. $Alice$ defines the contents of the contract and provides it to the users of the set R (formed by users Bob_i, for $i = 1$ to $i = n-1$, where n is the number of signers of the multiparty contract) to receive the signatures of Bob_i on the contract, for $\forall i$.
- Receiving Signer Bob_i. Recipient of the contract to be signed. These signers have to decide if they want to accept the text of the contract and whether to sign it or not.
- Smart Contract. Program deployed on the blockchain that is used to manage the contract signing process providing fairness to the exchange.

Both the proposing signer and the set of receiving signers interact with the smart contract using their blockchain addresses. In brief, the protocol has the following steps.

- $Alice$ decides whether she wants to sign a confidential contract or a public contract. Public contracts would use a simplified version of the protocol.

- $Alice$ defines the set of signers of the contract.
- The parties execute a blockchain-based exchange in order to sign the contract and to provide non-repudiation proofs. This exchange is performed on-chain executing the functions provided by a smart contract. The protocol ensures that the smart contract cannot access the contract when confidentiality is required, thanks to the use of an innovative technique that will be explained in this section

States have been used to manage the signature process. Figure 14.1 depicts the states for the confidential contract signing protocol. States *Rejected*, *Canceled* and *Finished* are final states. The first and the second represent aborted signature processes caused by different reasons. The last is the state that represents a completed signature process.

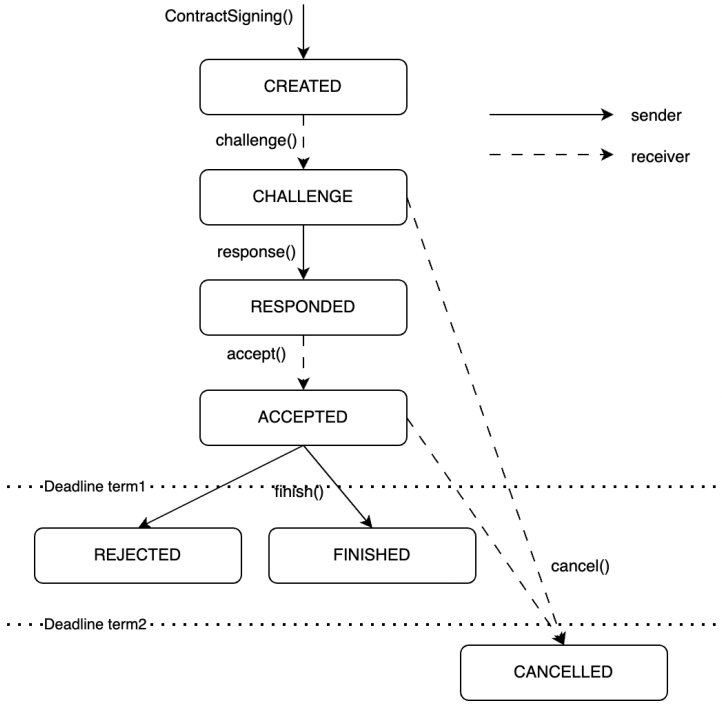


Figure 14.1: States of the multiparty contract signing protocol

The first phase of the contract signing operation, the *Creation* stage, is performed by $Alice$. Before executing this stage and in order to configure the protocol operational conditions the corresponding ECC cryptographic parameters must be published to ensure privacy and security according to the international standards [180]. These parameters must be:

- F_p: finite field of p elements, where p is a prime number.
- $E(F_p)$: elliptic curve on F_p.
- G: cyclic subgroup generator of points over $E(F_p)$ with order n.
- n: order of the subgroup G.

- h: cofactor of the subgroup generated by G, the same as the order of the elliptic curve divided by the order of the cyclic subgroup n (normally $\leqslant 4$).
- $P\mathbf{x}[b]$: product of a point P and the scalar b on $E(F_p)$.
- $a \leftarrow [1, n-1]$: $Alice$'s private key, randomly generated in $[1, n-1]$.
- $A = G\mathbf{x}[a]$: $Alice$'s public key.
- The users that interact with $Alice$ must verify that A is a valid point of the curve and that $A\mathrm{x}[h]$ is not a point in the infinite.

The notation that will be used for the definition of the protocol is summarized in Table 14.1.

The protocol provides a fair exchange of signatures on a confidential contract. Since the contract is confidential it cannot be stored in the blockchain. Moreover, the smart contract could not access the text of the contract in any case. The proposing signer $Alice$ initiates the exchange using the DApp to execute the first step of the protocol. In this step, the contract, previously encrypted, is registered on the blockchain executing a function of the smart contract. The encryption of the contract guarantees that all the signers, and only them, can access the contract.

After this step, the signers cannot decrypt the contract yet. To allow the signers to decrypt the contract, the smart contract includes two steps. The users will execute these steps to obtain the contract before they decide to accept or reject the contract. These steps follow the challenge-response formula. Using the response to the challenge, the smart contract verifies that all the signers can access the content of the contract, although it is not able to decrypt it.

Since the protocol allows the signature of multiparty contracts, the smart contract has to verify that every signer can decrypt the same contract. We use a pair of keys, that we call *Contracting Keys*. Each signer has its own pair of *Contracting Keys* before the signature of the contract.

After the decryption of the contract, each signer Bob_i has to accept and sign the contract executing the acceptance function provided by the smart contract, making a transaction on the blockchain. At the end, the proposer $Alice$ executes the last transaction. If all the members of R have accepted the contract, this transaction represents the signature of $Alice$ on the contract.

The protocol for the multiparty fair signature of contracts has several phases. The names of these phases are *Creation*, *Challenge*, *Response*, *Accept* and *Finish*. Moreover, we have included an additional phase, *Cancel*, that can be executed optionally by the receiving signers. These phases are described below using the notation included in Table 14.1:

1. **Creation**. $Alice$ generates her pair of contracting keys (a, A) and generates the random seed v that will be used to encrypt the contract M. If it is necessary, for this encryption operation, the content of the contract can be divided into fragments $(M[j])$.

   Starting with the contract signing, $Alice$ executes the function $creation()$ of the constructor of the factory of the smart contract deployed by the service provider.

Table 14.1: Notation of the multiparty contract signing protocol

Notation	
$Alice$	Proposing Signer.
R	Set of Receiving Signers.
Bob_i	Single receiving signer.
R_a	Set of receiving signers that have accepted to sign the contract.
M	Contract to be signed.
X, Y	Messages X and Y are concatenated.
$U \blacktriangleright e.f$	Function f of e are executed by U.
$\leftarrow [,]$	Randomly generated number in the interval.
$term1$	Timeout for Bob_i to accept the contract.
$term2$	Timeout for $Alice$ to finish the exchange.
D	Deposit.
a	$Alice$'s contracting secret key.
A	$Alice$'s contracting public key.
b_i	Bob_i's contracting shared secret key.
B_i	Bob_i's contracting public key.
G, p, n	System cryptographic parameters.
v	Encryption secret nonce used by $Alice$.
s_i	Encryption secret nonce used by Bob_i.
$V = G\text{x}[v]$	First element of the ZKP proof.
$M[i]$	Fragments of contract M to be encrypted, in the range $0 < M_i < length$.
$key = random.seed(hash(v))$	Encryption key.
$hash()$	Hash function.
$C[i] = M[i]$ XOR key	Encrypted contrat.
c_i	Challenge sent by Bob_i to $Alice$.
r_i	Response to the challenge c_i.
Z_{i1}=$G\text{x}s_i$	Encryption of the receiving signer shared secret contracting key. First Part.
$Z_{i_2} = (A\text{x}[s_i]) \oplus (b_i)$	Encryption of the receiving signer shared secret contracting key. Second Part

This new instance will be useful for the management of the new contract signature. The execution of the function $creation()$ includes the following parameters: the set of receiving signers R, $Alice$ commitment to send the correct key V, the ciphered element C, the terms for the signature of the contract $\{term1, term2\}$ and the remaining cryptographic parameters. The first term $term1$ determines the acceptance period for the receiving signers, while the second term $term2$ specifies the term for the proposing signer to finish the signature of the contract, that is, the instant since when the signers will have access to the signed contract together with the non-repudiation proofs. If the proposing signer has not completed the exchange by $term2$, the remaining signers can obtain a cancellation proof executing the function $cancel()$ of the smart contract. Optionally, a deposit D or payment for the service can be included in this stage.

We have performed an adaptation of the encryption method of the Elliptic Curve Integrated Encryption Scheme (ECIES)[38], where the element V is not used by the receiving signers to decrypt the contract. Instead, V is used as a commitment by $Alice$ of emission of the right key that will be checked by the smart contract through a Zero-Knowledge Proof (ZKP) in the third phase of this protocol (*Response* subprotocol). In this way, the smart contract verifies on-chain that all signers will be able to decrypt the same plain text of the contract without accessing the actual content of the contract. The ZKP performed to achieve this feature of our proposal is a customized version of the Identification Scheme proposed by Schnorr and translated to ECC [181, 46]. In general, an identification scheme is a cryptographic tool for a user to prove his/her identity to other parties. This user is often called *prover* and the parties that check the identity are called *verifiers*. In our case, the *prover* is the party who proposes the contract to be signed and the deployed smart contract plays the role of a *verifier*, which checks and gives evidence that the signed contract is the same for all signers (i.e. all signers can get access to the same key to enable the decryption of the encrypted contract).

2. **Challenge**. In a multiparty contract signing scenario, each signer can choose individually if he wants to sign the contract or reject it. In order to sign the

Subprotocol 21: Creation

1. $Alice$: generates $M, a \leftarrow [1, n-1], A = G\text{x}[a], v \leftarrow [1, n-1]$
2. $Alice$: $key_0 = random.seed(hash(v))$
3. $Alice$: encryption of C.
   If it is required, M can be fragmented in blocks.
   $V = G\text{x}[v]$
   FOR $j = 1$ **TO** $M.length$
   $C[j] = M[j]$ XOR key_{j-1}
   $key_j = random.seed(hash(j||v))$
4. $Alice \blacktriangleright SM$:
   $SM.creation(Alice, R, V, C, term1, term2, A, G, p, n, D)$
5. SM: $State_i = Created, \forall i$

contract, each signer Bob_i has to accept and sign the contract before the deadline $term1$. Before the acceptance, $Alice$ allows access to the contract to all $Bob_i's$ while keeping it confidential. For this reason, the text of the contract cannot be stored in the blockchain, so it cannot be part of a transaction. $Alice$ generates the elements to deliver, in a confidential way, the key that will be used to decipher the contract, (that is, the proposing signer sends the key that has been committed to send at the beginning of the exchange, at it is explained in Subprotocol 23) and also that the smart contract can ensure that a signer Bob_i can access the contract. Signer Bob_i generates its own pair of contracting keys b_i, B_i and uses its public contracting key b_i, a variable that will be used as a challenge c_i, and its contracting shared secret key encrypted with the public contracting key of the proposer $Alice$ $\{Z_1, Z_2\}$ as parameters when Bob_i executes the function *challenge()* of the smart contract.

The smart contract then checks that $term1$ has not expired, that the state for each signer is *created* and that the identity of the executor of the function, Bob_i is a member of R. If all the checks stand, the state for the signer is set to *Challenge* and the signer, Bob_i, is added to the set R_a.

Subprotocol 22: Challenge

1. Bob_i : generation: $b_i \leftarrow [1, n-1], B_i = G\text{x}[b_i], s_i$
2. $Bob_i : Z_{i_1} = G\text{x}[s_i], Z_{i_2} = (A\text{x}[s_i]) \oplus (b_i)$, $c_i \leftarrow [1, n-1]$
3. $Bob_i \blacktriangleright SM.challenge(Z_{i_1}, Z_{i_2}, B_i, c_i)$
4. SM:
   IF$(now < term1)$ AND $(Id == Bob_i)$ AND $(State_i == Created)$
   $State_i = Challenge$
   Add Bob_i to R_a

3. **Response**. The proposing signer $Alice$ has to allow access to the contract to be signed by all the members of R, keeping it hidden from the rest of the world. For this reason, the content of the contract cannot be included in any transaction on the blockchain. This situation leads us to a solution in which $Alice$ generates the elements that are required to guarantee access to the contract, taking into account that the protocol has to ensure that the key that all the signers, members of R_a, obtain is the same. This way all the receiving signers will sign the same contract. That is, $Alice$ has to send the key that was committed during the phase $Creation$, using parameter V.

   The smart contract has to guarantee that all the signers R_a will have access to the same decrypted element. With this objective, in phase 2, $Challenge$, each signer Bob_i has generated his pair of contracting keys (b_i, B_i) and has provided to the smart contract his public contracting key B_i, together with a nonce c_i used as a challenge for the execution of a Zero-Knowledge Proof used by the smart contract to check that all the receiving signers can receive the key that $Alice$ had hidden in V. Moreover, the signer Bob_i encrypts his contracting secret shared key resulting in $\{Z_{i_1}, Z_{i_2}\}$ using $Alice$ public contracting key. This way $Alice$ can recover the

contracting secret shared key and send a response to Bob_i to open the contents of the contract. During the execution of this subprotocol, the signers can obtain the contracting secret shared keys of other signers. However, after this step, the contracting shared secret key is never used again by any signer, since the final acceptance of the contract is made by a regular function call to the smart contract, so the knowledge of this key does not lead to anything else that the content of the Contract, that all signers already have using its own contracting shared secret key. Then, no further information is leaked that could lead to security breaches.

To answer the challenge, the proposing signer generates for each signer a response to the challenge in the form of a ZKP using the secret element used in the encryption of the contract in the first step of the protocol, the challenge, c_i, and Bob_i's contracting shared secret key. The Smart Contract will store the parameters. With this response, the Smart Contract can verify, by means of the stored data, that the signer Bob_i will be able to know the secret element that will allow him to decipher the contract. However, the Smart Contract will not know this element and therefore the contract will remain confidential.

The performed ZKP allows the smart contract to verify the commitment using the secret key v performed by $Alice$ in phase 1, *Creation*. This way $Alice$ can prove to the Smart contract that she is sending the appropriate secret element v to every receiver signer, without revealing its value, because the response to the challenge c_i sent by any signer is coherent with the value committed publicly in V. If the verification stands, the smart contract concludes satisfactorily the process and sets the state to $Responded$.

In the last step of this phase, the signers' members of R can isolate v from r_i due to their knowledge of b_i and then they will be able to read the contents of the confidential contract to be signed.

Subprotocol 23: Response

1. $Alice$: decrypts: $b_i = Z_{i_2} \oplus (Z_{i_1}\text{x}[a])$
2. $Alice$: computes: $r_i = v - b_i * c_{i\ mod n}$
3. $Alice \blacktriangleright$ SM.response(r_i)
4. SM:
   IF ($Id == Alice$) AND (($now < term1$) OR ($R_a == R$))
      FOR ($\forall Bob_i \in R_a$)
         IF $V == G\text{x}[r_i] + B_i\text{x}[c_i]$
            $State_i = Responded$
         Deposit D is refunded to $Alice$
5. Bob_i:
   $v = r_i + b_i{*}c_{i\ mod n}$
   $key_0 = random.seed(hash(v))$
   FOR $j = 1$ **TO** n
      $M[j] = C[j]$ XOR key_{j-1}
      $key_j = random.seed(hash(j||v))$

4. **Accept**. In a multiparty contract signing scenario, each signer, Bob_i, has to sign the contract using function *accept()* before deadline $term_1$. The state of this signer will be changed to $State_i = Accepted$. If a signer Bob_i does not sign the contract before $term_1$, it is assumed that he doesn't want to sign the contract and the state for this signer will be $State_i$==*Rejected*.

   Since it is required that all signers sign the contract for this to be valid, the protocol will not accept partially signed contracts, that is R_a has to be the same set R, for the contract to be signed.

Subprotocol 24: Accept

1. $Bob_i \blacktriangleright SM.accept()$
2. SM:
   IF($now < term1$) AND ($Id == Bob_i$) AND ($State_i == Responded$)
   $State_i = Accepted$
   Add Bob_i to R_a

5. **Finish**. To conclude the contract signing process, before the deadline defined by $term2$, $Alice$ must also sign the contract. As it has been said before, in a multiparty contract signing operation all signers must sign the contract. For this reason $Alice$ checks the state of all the receiving signers using the function *finish()* of the smart contract. $State_i$ must be $Accepted$ for all the signers Bob_i.

   The smart contract realizes three verifications:

   - The identity of the user who executes the function, that can be only the proposing signer, $Alice$.
   - The instant of the execution of the function, that has to be between $term1$ and $term2$.
   - The acceptance of the contract. All the receiving signers must have accepted the contract. The state for these signers should be $Accepted$, and they are included in R_a. IF $R == R_a$, then all the receiving signers have signed the contract.

   If the three verifications stand, then $Alice$ can finish the exchange before $term2$. Now the contract has been signed by all the parties.

   Ultimately, if all parties sign the contract, that is, if $State_i == Finished\ \forall i$, both the proposing signer $Alice$ and the members of R will have access to the signed contract through WEB3 or similar interface. However, if $Alice$ does not execute satisfactorily the *finish()* function the signers can execute the optional phase 6, $Cancellation$.

6. **Cancellation**. This function can be executed by the signers Bob_i, if the proposing signer A does not execute the *finish()* function during its execution period when the receiver signer has accepted the contract. This step is optional. If the proposing signer $Alice$ follows the protocol and executes *finish()*, then the state for the

Subprotocol 25: Finish

1. $Alice$ ▶ SM.finish( )
2. SM:
   IF ($Id == Alice$) AND (($term1 < now < term2$) AND ($R_a == R$))
      FOR ($\forall Bob_i \in R_a$)
         $State_i = Finished$
      ELSE ($\forall Bob_i \in R_a$)
         $State_i = Rejected$
   Deposit $Alice$ receives the refund of deposit D.

Subprotocol 26: Cancellation of Acceptance

1. Bob_i ▶ SM.cancel()
2. SM:
   IF($now >= term2$, $Id == Bob_i$ AND $State_i == Accepted$)
      $State_i = Canceled$

receiving signer that executes the function would be either *finished* or *rejected*. These states are the final states of the protocol, as can be seen in Figure 14.1. Only if the state for Bob_i is *accepted*, meaning that finish() has not been executed, then the state of Bob_i will be changed to the value *canceled*.

In this phase, the smart contract makes three verifications: the identity of the receiving signer, the state of the exchange ($Accepted$, meaning that the proposing signer has not finished the exchange) and the deadline (this phase can only be executed after $term2$).

The smart contract makes a transaction to cancel the exchange. In this case, if the state is $Accepted$ for all the receiver signers, the proposer $Alice$ will not receive the refund of the deposit. Thus, this deposit is used to motivate $Alice$ to finish the exchange before the deadline.

14.4 Implementation

The implementation of the protocol is useful to prove that the costs related to the execution of the functions of the smart contracts are small enough for the protocol to be viable. For the implementation of the proposed protocol we have used the Ethereum blockchain and the smart contract has been programmed using Solidity.

Instead of using an isolated smart contract for each contract signature using the protocol, we have deployed a Smart Contract that will be used to deploy the smart contracts for the signature of confidential contracts. This matches the factory method programming pattern to deploy new instances of a smart contract to process each new contract signature. The Factory Contract deploys new smart contracts for each signature process. Thanks to the use of this pattern we obtain reusability of the code and easy access to the generated contract signatures reducing execution costs.

We use several data structures to keep the variables of the protocol, such as the group of signers, where an array is declared to store their addresses. Moreover, the

smart contract has to maintain the value of the state of the exchange for each signer, thus a mapping structure has been used to keep track of these states, for every signer, together with the values used in the implementation of the Zero-Knowledge Proof.

Moreover, we have also used variables of the smart contract to keep the timeouts used in the protocol ($term1$, $term2$). Another variable stores the blockchain address of the proposing signer $Alice$. In order to manage the time we have declared a variable called *start* to keep the current time of the instant a contract signature process is initiated, so the timeouts $term1$ and $term2$ are calculated using this value.

Implementing cryptographic operations in solidity could be challenging. If the implementation of a protocol is not efficient enough, it will have high execution costs, both in terms of time and gas consumption. For this reason, we have used Elliptic Curve Cryptography (ECC) to specify and implement the protocol. In this way, the implementation of our protocol uses 256-bits ECC that offers a security level of 128 bits according to the NIST recommendation [35]. Therefore, to achieve the same level of security by using RSA-like cryptosystems (as in the protocols described in Chapter 8, Chapter 9, and Chapter 13), the variables used in the implementation should have a length of 3072 bits that will result in poor performance due to the high cost of the modular exponentiation operation over 3072-bit numbers.

We have chosen the same ECC settings as ECDSA in [170] for our pilot study of the protocol implementation. In order to implement the ECC operators, such as scalar multiplications and additions, we have used the library published by Jordi Baylina in [182]. Also, the benchmark of Baylina's library[1] outlines the gas cost of using this library in ROPSTEN network.

Also, we have checked the time consumption for each local operation in order to know the computational time of the isolated ECC operations used in the protocol. The addition of two points in the curve takes 110ms on average and the scalar multiplication takes 180ms on average in an Intel Core i5 CPU computer running a Mac OS 11.3.1 Big Sur operating system with RAM of 16Gb. Therefore, the delay introduced by the performance of the operations in relation to the waiting time of each transaction is negligible. The use of 256-bit numbers instead of 3072-bit not only saves time but also saves storage space and, thus, money (according to the Ethereum Yellow Paper [19] the fee is 20,000 Gas to keep each 256-bit word).

Most smart contract functions contain ordinary tasks, such as checking timeouts or executing them by authorized accounts. The most challenging function of the development of the smart contract is the *response*() function, where it must be verified that Bob_i will be able to know the secret element that will allow him to decipher the contract.

This verification will be done by the smart contract, which will perform the ZKP to verify that $Alice$ has sent the appropriate secret key v in the creation phase, without revealing its value, checking the response r to the c_i challenge sent by Bob_i. The verification consists in checking the equality between the V point of the elliptic curve sent by $Alice$ at creation, and the result of this expression G x $[r_i]$ + B_i x $[c_i]$. If this verification is accomplished, the smart contract will save the response r_i, and will set the receiver's state to $responded$. Listing 32 depicts the $response$() function.

[1] https://github.com/jbaylina/ecsol/pull/8

```
function response(address _receiver, uint256 _r) public {
    uint256 Grx;
    uint256 Gry;
    uint256 Bcx;
    uint256 Bcy;
    require(
        block.timestamp < start + term1,
        "The timeout has been reached"
    );
    require(
        sender == msg.sender,
        "Who try to response the challenge is not the original
            proposer"
    );
    require(
        receiversState[_receiver].state == State.challenged,
        "The receiver is not in the challenged state"
    );

    // Check V == G x [ri] + Bi x [ci]
    (Grx, Gry) = deriveKey(_r, gx, gy);
    (Bcx, Bcy) = deriveKey(
        receiversState[_receiver].c,
        receiversState[_receiver].bx,
        receiversState[_receiver].by
    );
    require(vx == Grx + Bcx, "V and Gx[ri]+Bix[ci] are not equals");
    require(vy == Gry + Bcy, "V and Gx[ri]+Bix[ci] are not equals");
    receiversState[_receiver].r = _r;
    receiversState[_receiver].state = State.responded;
}
```

Listing 32: response() function

The implementation of these smart contracts has been done with the Hardhat Ethereum development environment, and using the Ethers.js, a library that lets us interact with the Ethereum Blockchain and its ecosystem.

14.5 Security properties analysis

The security analysis of the presented protocol analyses the desired properties for electronic contract signing systems, including confidentiality.

1. **Effectiveness.**

   The presented electronic contract signing protocol is effective. Therefore, all parties will have access to the signed contract if they behave in accordance with the protocol. In order to sign a new contract, the proposing signer creates a new instance of the smart contract. If all parties execute the step according to the protocol correctly ($challenge()$, $response()$, $accept()$ and $finish()$), they will have access to the proofs of the signature of the contract. This can be easily deduced from the protocol presented in section 14.3. After the exchange, all the

signers will have non-repudiation proofs of having signed the contract from all the signers.

2. **Fairness and Transferability of Evidence**

   The contract signing protocol provides strong fairness [168]. After the execution of the protocol, all the parties have received the appropriate element or, if it is not the case, neither party has received any useful data on the element from the other party, which provides great impartiality.

   In addition to this, the proofs generated by the protocol could be transferred to any external arbiter to check the final state of the exchange, without the need of any further verification.

   According to the protocol, any signer Bob_i signs the contract if he makes a transaction to accept the proposal ($State_i == Accepted$). The execution of the function *accept()* is considered the non-repudiation of acceptance proof for the signers since this transaction leads the exchange to the state $Accepted$. Moreover, the proposing signer will not receive the non-repudiation evidence of the signature of the contract generated by the smart contract, unless she makes a final transaction, after allowing the recipients to decrypt the contract (allowing the smart contract to verify the provided decryption key), to finish the exchange signing himself the contract ($State == Finished$).

   If any party misbehaves and does not follow the steps of the protocol specifications, that is, if it does not execute the functions $accept()$ or $finish()$, then the smart contract guarantees a fair state for all users without the need for TTP intervention:

   - $accept()$ not executed. The smart contract will set its status to $Rejected$, after the timeout, if any signer has not executed the $accept()$ function.
   - $finish()$ not executed. Signers can execute function $cancel()$ to finish the exchange in a fair way, if the proposing signer has not executed function $finish()$. Then the smart contract will change the final state of the exchange to $Canceled$.

   The protocol does not allow any circumstance in which a signer can obtain contradictory evidence because the state of the exchange is actualized in the smart contract (for this reason, there is no action that may lead to a situation of weak fairness [168]).

3. **Time Parameters**

   The time parameters to consider are $timeliness$ and $timestamps$. For the proposed protocol a successful signature will always be completed before the *term2* deadline. If the signature is not successful, we have different deadlines depending on how the exchange was performed. If a signer does not accept the contract, the signature process will be classified as *Rejected*. If after the acceptance of the contract by the signers, the proposing signer does not finish the exchange, then the exchange can be canceled at *term2*. In addition, the system motivates the sender to complete the exchange before *term2*. This motivation is

done through a deposit that is locked in the smart contract. The deposit will be returned to the sender in case of completion before *term2*.

Keep in mind that the blockchain marks the time of all the transactions carried out on it, so the exchange has timestamps for all operations on the smart contract that generate transactions, such as the phases of the proposed protocol.

4. **Non-repudiation**

   The contract signing protocol must provide evidence of non-repudiation of signature. Regarding the origin, the proposing signer cannot deny having executed the $creation()$ function since there is a transaction in the blockchain from her address that contains the addresses of the signers and the encrypted contract. The execution of the function *accept()* is considered the non-repudiation of acceptance proof for the signers since this transaction leads the exchange to the state $Accepted$. In addition, the transaction related to function $response()$ proves that the proposing signer has provided the key to decrypt the contract to all signers. The function *finish()* is considered the non-repudiation proof for the proposing signer since this transaction leads the exchange to the state $Finished$.

   Regarding the receiving signers, each signer Bob_i cannot deny having signed the contract because there is an acceptance transaction from his address stored in the blockchain. This transaction is the signature of the signer on the text of the contract deciphered after the execution of $response()$. The smart contract validates that all the signers have been able to decrypt the contract and also that all the signers have signed the same contract.

5. **Confidentiality**

   The signed contract is confidential, only the signers can access its content. For this reason, the contract cannot be included in clear in any blockchain transaction and cannot be stored in the blockchain. This way the contract cannot be included as a parameter in the functions of the smart contract and also we have to prevent the smart contract could obtain the decryption key. However, the protocol has to verify that the contract obtained by all the signers is the same.

   In the first step of the protocol, the proposing signer executes the function to create the contract signing operation, including the encrypted contract (C_1 and C_2). Then each signer provides a way for the proposing signer to send the decryption key privately, using elements Z_1 and Z_2. in a Zero-Knowledge Proof. The smart contract verifies that the proposing signer has provided the right decryption key, that can be obtained from the response to the challenge.

   No other entity is involved in the procedure and the transactions on the blockchain are limited to the encrypted contract and some hidden decryption keys, so the contract will be confidential.

14.6 Performance analysis

Once we have finished the implementation of the proposal, we have tested the results with the help of the $Hardhat$ development environment, which has some useful plug-

ins like $hardhat-gas-reporter$. With this plug-in, we have a Gas usage per unit test, that uses metrics for method calls and deployments.

We have performed some tests to determine the efficiency of the system in terms of gas cost since the economical execution costs could be a concern in the development of this service. We have used the local Hardhat network, a personal blockchain used for Ethereum development, to isolate the performance conditions and possible problems of a real network like the main Ethereum or the Rinkeby test networks.

From the results of the tests, we have detailed the gas cost of the functions, executing the protocol with two signers. In Figure 14.2 we can see the gas cost of the main functions of the protocol, according to the *Hardhat-gas-reporter* plug-in. The cost of this analysis is computed in gas, but to know the exact price of these transactions we also need the gas price, set in Ethers (ETHs), and the Ether to US-Dollars exchange rate, but neither one nor the other is fixed. The cost can also be improved using different Ethereum Virtual Machine (EVM) compatible networks, like Binance Smart Chain (BSC), that uses Binance Coin (BNB) cryptocurrency coins.

Methods		
Contract	Method	Avg gas cost
ConfidentialContractSigning	accept	52997
ConfidentialContractSigning	cancel	26654
ConfidentialContractSigning	challenge	211471
ConfidentialContractSigning	finish	46471
ConfidentialContractSigning	response	5166884
ConfidentialContractSigningFactory	createDelivery	1461696
Deployments		
ConfidentialContractSigningFactory		1767710

Figure 14.2: Gas cost of the multiparty contract signing smart contract

If we analyze the gas cost of the functions and deployments of these smart contracts, we can notice that the more expensive functions in terms of gas cost are the deployment of the Factory smart contract, the deployment of the $ContractSigning$ contract via the $createDelivery()$ function of the factory, and, above all, the response() function, where the elliptic curve related calculations are performed.

14.7 Conclusions

This chapter presents a complete confidential multiparty solution for the electronic contract signing service using blockchain-based fairness, based on the two-party solution introduced in the previous chapter. The protocol manages the exchange of signatures among signers and guarantees the fairness of the exchange without the involvement of any TTP to guarantee the required properties since the actions of the signers are stored in the blockchain and, in case any signer does not follow the protocol, the smart contract will generate the corresponding evidence to preserve fairness. Although the presented solution is a multiparty protocol, the execution costs will be kept controlled since the number of interactions among the parties is not directly related to the number of signers.

Since the contracts are confidential, only the signers can access the text of the contract. This way the smart contract must manage the exchange and guarantee fairness without being able to access the contract. In terms of protocol, this means that the contract must be encrypted, and the smart contract, without having access to the decryption key, must ensure that all recipients can decipher the same contract. To achieve this property we have included in the protocol a challenge-response stage and a Zero-Knowledge Proof. This way, the smart contract can verify that all the signers have access to the contract and that the contents of the contract that the users can decrypt and sign are the same for all of them. Moreover, the involved parties can be certain that the protocol will be completed at a certain finite point in time. The subprotocols have been designed using Elliptic Curve Cryptography operations.

The transactions on the blockchain and the state of the contract signing instance are proofs of the final state of the exchange. These proofs can be transferred and checked by any validator. This way, the signers cannot deny having signed the contract. A validator can check that the provided contract is the contract that was signed using the protocol in a precise instant of time, thanks to the timestamp of the blockchain blocks.

Regarding the implementation, we have used smart contracts that are able to manage more than just one contract-signing operation. We have not included in this chapter the details of the smart contracts and the implementation of the ZKP. However, some hints about the most relevant operations are depicted in section section 14.4. The implementation is useful to analyze the performance of the protocol in terms of cost since an inefficient implementation would result in high execution costs in terms of time and gas consumption. To achieve all the desired properties, including confidentiality, the protocol uses advanced cryptography. The use of ECC allows the use of shorter parameter lengths than those used in previous proposals for e-commerce protocols using blockchain, guaranteeing the same level of security. This allows controlling the costs associated with the execution of smart contracts.

CHAPTER 15

MICROPURCHASES USING PAYMENT CHANNELS PROTOCOL

After examining certified notification protocols and contract signing methods, we now turn our attention to the use of blockchain for developing a micropurchases protocol. As described in [121], a new, efficient, and secure micropurchases scheme was proposed to facilitate low-value transactions while ensuring customer privacy. In this chapter, we propose an enhanced version of this protocol by utilizing blockchain technology. Unlike the previous system mentioned in [121], which employed specific currencies for each dealer, our new system leverages cryptocurrencies to enable a fair exchange between customers and dealers for goods or services. Furthermore, our proposal eliminates the need for banks to manage customer and dealer accounts by implementing smart contracts directly on the blockchain.

Our system establishes a payment channel that not only prevents double spending and overspending but also protects against falsification. It also enables customers to securely request refunds for unused amounts. Dealers, on the other hand, can collect payments even before closing the channel. Additionally, the channel's functionality includes redirection, facilitating the transferability of cryptocurrencies in what is called a *multihop* solution. For more detailed information on micropurchases and micropayments, please refer to section 5.3.

15.1 Contribution

This work significantly advances the current understanding and implementation of blockchain-based micropurchases systems by introducing a novel protocol that addresses several key issues inherent in previous systems. Firstly, our protocol eliminates the need for Trusted Third Partys (TTPs), which are commonly required in traditional micropurchases systems to manage and verify transactions. By deploying smart contracts on the blockchain, our system autonomously manages transactions between parties, ensuring integrity and reducing potential points of failure or fraud.

Secondly, the protocol introduces a multihop payment channel feature, which allows the transfer of funds across multiple parties without necessitating a liquidation after each transaction. This capability not only enhances the fluidity of transactions but also considerably reduces transaction fees, as fewer transactions are recorded on the blockchain. This feature is particularly innovative, as it supports a dynamic and flexible use of payment channels that can adapt to various user needs and transaction contexts.

Moreover, our protocol emphasizes security and privacy, implementing mechanisms to prevent double spending, overspending, and falsification of transaction records. It also ensures that all transactions within the payment channel are confidential and only accessible to the parties involved. These security measures are critical in maintaining trust among users and ensuring the viability of the system for real-world applications.

In addition to these technical contributions, the protocol also introduces an efficient way to handle refunds and channel redirections, further enhancing the flexibility and user-friendliness of the system. These features make it possible for users to manage their funds more effectively, opting to redirect or refund their money as needed without cumbersome processes or the involvement of intermediaries.

Comparing this protocol to the Lightning Network [129], there are both similarities and differences in their architectures, functionalities, security mechanisms, and efficiencies. Both systems utilize payment channels to facilitate off-chain transactions and reduce blockchain costs. Our micropayments protocol uses μ-coins and proof elements to prevent counterfeiting, double spending, and overspending, with a focus on specific security and privacy needs. It supports multihop transfers, enhancing efficiency without requiring liquidation after each use. In contrast, the Lightning Network, a second-layer Bitcoin solution, employs Hashed Timelock Contracts (HTLCs) to ensure secure, fast, and low-cost transactions across multiple nodes, with automatic payment route selection and continuous security monitoring. Both aim to minimize blockchain congestion and transaction costs, but our protocol offers tailored solutions for specialized environments, whereas the Lightning Network is designed for broader applicability within the Bitcoin ecosystem.

Overall, the contributions of this protocol are multifaceted, addressing technical, operational, and user experience aspects of blockchain-based micropayments. By offering a solution that is both technologically advanced and aligned with user needs, this work sets a new standard for the implementation of micropayment systems using blockchain technology.

15.2 Protocol design

The protocol is made up of different phases. In the first one, the configuration of the system and the generation of keys are carried out. Next, the buyer user selects the business in which he wishes to make purchases and requests the service or product to be purchased. In the next phase, the opening of the channel is carried out. Once the channel is open, purchase operations can be carried out, which include payment and delivery of the purchased product or service. Finally, the system has the channel

payment or liquidation phase or, alternatively, the channel transfer phase (multihop exchange), where the channel is converted for the purchase operation with a new user.

The actors involved in the purchase protocol are the buyer C and the seller M. In addition to these, the protocol is regulated by a smart contract (SC) deployed on the blockchain that regulates operations on the open payment channel between C and M, saving the exchange data on the blockchain. As it is a system of transferable channels, multiple vendors can also be involved. To define the transfer of the channel, the second seller will be called N.

Table 15.1 shows the notation used in the protocol description.

Table 15.1: Notation of the Micropurchases using payment channels protocol

Notation	Description
C	Buyer
M	Seller
N	Seller on a transferred channel
SC	Smart Contract
S_{id}	Service identifier
Γ	Channel Element Array
W_{LC}	Chain Generator
W_{0C}	Chain identifier
W_{0M}	Identifier for chain liquidation
c	Number of μ-pay coins.
v	Value of μ-coins
T_{Exp}	Expiration Date
Δ_{TD}	Deposit Period
Δ_{TR}	Redemption period.
K_s	Session key
$E_{Ks}[M]$	Symmetric Encryption of the message M with the key K_s
$Channel_{Id}$	Channel identifier
$H()$	Hash Function
Q	Amount associated to the payment channel

We describe the payment protocol in its different phases: *Service request* by a client, *Opening* of the payment channel, *Purchase* (exchange between payment for a product or service), *Liquidation of the payment channel* and, finally, *Transfer of the payment channel* for other uses (*multihop*). All these phases, with the exception of the *Buy* phase, are carried out with *onchain* communications. This means that the actions carried out by the different actors are carried out through function calls to the smart contract deployed to regulate the payment channel. This smart contract will control that each actor can only carry out the actions that correspond to them and will record them in the blockchain.

Each of the phases of the proposed protocol is described below:

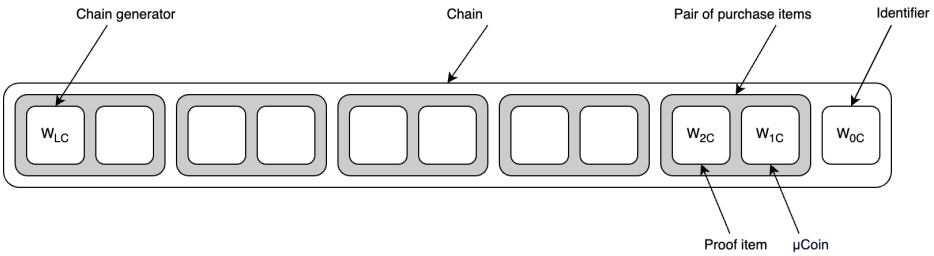


Figure 15.1: Structure of the chain of purchase items

Service Request

In this phase, the client must select a service offered by the different vendors. Therefore, each seller M has published its list of available services, S_{Idi}. Then C selects a vendor M and a service from the list of services it offers. The selected service is identified by S_{id}. The client decides how many μ-coins c wants to use in the channel. Each μ-coin will have a value v that can be determined based on the price of the selected service (the price of S_{id} will be a multiple of v). The client sends this data to M which generates W_{LM} (where $L = 2c+1$) and hashes this item L times $W_{0M} = H^L(WLM)$. Seller M transmits W_{0M} to C so that it can open the payment channel for this service. It should be noted that this chain is generated to allow partial liquidations of the channel. If only one liquidation is allowed, then it would be enough for M to apply the hash function once on the generator to obtain the identifier.

Channel Opening

C proceeds to open the channel for micropayments by transferring the amount $Q = c * v$ to the smart contract. Remember that c is the number of μ-coins that you want to deposit in the channel and that v is its value. It performs a transaction balance check of C and stores the amount Q.

Then C generates W_{LC}, where $L = 2c+1$ and hashes it L times to get W_{0C}. We will call $W_{(L-1)C}$ the result of hashing W_{LC}. The last element of the chain will be called W_{0C}. Within this chain, the elements with odd subscript will represent μ-coins while the elements with even subscript will represent the proof of payment of the previous μ-coin. It has been decided that the chain of M has the same length as the chain of C to simplify the notation although a chain with half the elements would be enough since the chain of M does not contain payment items intercalated with proof items.

T_{Exp} is the expiration date of the channel, Δ_{TD} defines a period after the expiration date in which the channel content can be transferred to the account of the payment recipient or the transfer of the channel to a new seller, while Δ_{TR} represents a period for the reimbursement of the remaining money, not used for payments in the channel, by C and for a possible transfer of the channel by part of M (see Figure 15.2).

Buyer C generates the element $\Gamma = (W_{0M}, S_{Id}, 2c, v, T_{Exp}, \Delta_{TD}, \Delta_{TR}, W0C)$ with these elements. Subsequently, the buyer C calls a function of the smart contract to publish the element Γ on the blockchain and, in this way, make the creation of the payment channel effective. Additionally, the counter $j = 0$ stored in the smart contract symbol-

izes the count of disclosed secrets associated with the transaction of the μ-coins. We will identify a specific channel by $Channel_{Id} = H(\Gamma)$.

Purchase (Payment and Product/Service Exchange)

As many payment operations can be carried out as $\mu-$coins the channel contains before T_{Exp}. For this reason, M, through the smart contract, will check before each payment that the current date is less than the expiration date.

The purchase process is made up of three steps, forming an equitable exchange between the $\mu-$currency and the product or service. In each purchase operation, one or several $\mu-$coins can be used, so that the sum of their values is equal to the amount to be paid in the purchase operation.

The fair purchase operation consists of three steps that are executed off-chain, between C and M: sending the $\mu-$coins, sending the product or service and sending the proof of payment. $\mu-$coins will be revealed in reverse order of their creation in the hash chain. For this process, a secret session key K_s shared by C and M will be used.

1. Step 1.

   C sends M the message $m_1 = [E_{Ks}[W_{iC}], Channel_{Id}]$. Upon receiving the message, M decrypts W_{iC}, and checks the date and the channel identifier, that is, it checks that the channel is open in the smart contract.

   Next, the seller M verifies that there has been no reuse of μ-coin, verifying that $i > j$, where i is the order number of the μ-coin used in the payment and j the order number of the proof of the last μ-coin used. Also, M verifies that W_{iC} belongs to the chain $H^{i-j}(W_{iC}) == W_{jC}$. If it is the first payment made in this channel, the verification will be done as follows: $H^i(W_{iC}) == W_{0C}$. After these checks are complete, M saves S_{Id}, $Channel_{Id}$, W_{iC}, and $(j = i)$.

2. Step 2.

   M sends the product or service via the message m_2=$E_{Ks}[service/product]$. Upon receiving the message, C decrypts m_2 and prepares the proof attached to the μ-coin.

3. Step 3.

   C sends the proof associated with μ-coin in encrypted form, together with the index corresponding to its position in the chain. $m_3 = E_{Ks}[W_{(i+1)C}, (i+1)]$. Seller M decrypts and verifies $W_{(i+1)C}$. This verification is done by applying a hash function on the proof and it is verified that the result corresponds to the μ-coin

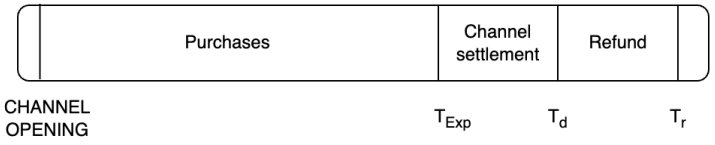


Figure 15.2: Payment channel life cycle of the micropurchases protocol

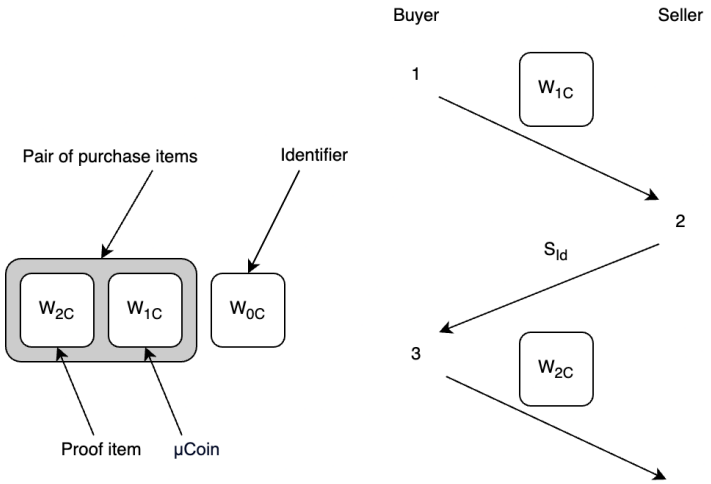


Figure 15.3: Purchase sub-protocol

used for the payment W_{iC}. Finally, M stores the value $W_{(i+1)C}$ and updates $j = i+1$.

Liquidation of channel

In order for the funds associated with the purchase operations to be transferred to M, it must perform the collection operation. This operation can be carried out once all the μ-coins associated with the channel have been used. It can also be performed if M has received one or several μ-coins associated with the payment channel, even if it has not been fully utilized. In an extreme case, M could charge for each μ-individual coin spent. For this, M must reveal each of the W_{iM} in reverse order of their creation. (It should be noted that the purpose of payment channels is to reduce costs and therefore the efficiency of the system is achieved through the simultaneous collection of different $\mu-$coins).

This operation is carried out on-chain and for this M accesses the smart contract, which will carry out the transfer unifying the amount associated with the μ-coins revealed. In the first step, the smart contract will verify that the current date is in the correct period. The collection period is between T_{Exp} and $T_{Exp} + \Delta_{TD}$. However, M can also execute the collection function before T_{Exp} to carry out a partial collection with the μ-coins received (in this scenario, additional payments can be made subsequently usins μ-coins of the channel). When executing the smart contract collection function, M will use the values of $Channel_{Id}$, W_{kM}, W_{kC}, k, being k as parameters. the index of the last received microcoin (or of the last coin to be charged).

The smart contract verifies that:

- $k > j$. This verification prevents the reuse of microcurrencies.
- $W_{jM} == H^{k-j}(W_{kM})$. This check makes it possible to verify that the user who executes the function is the recipient of the payment channel.

- $W_{jC} == H^{k-j}(W_{kC})$. It is verified that the μ-coin belongs to this channel.

Once all the verifications have been carried out, the smart contract transfers the balance associated with the micropayments to M. To do this, it determines the value based on the amount of μ-coins transferred and the value of each of them established in the channel. The number of μ-coins transferred is determined from the index provided by M. If the index is even, the value of $(k-j)/2$ will be transferred, while if it is odd, the value of $(k-j-1)/2$ will be transferred, since it is taken into account that the indices include the μ-proof coins. After this operation, the smart contract has to update the index $j = k$.

Channel Transfer

The protocol has been designed taking into account the interest in achieving transferable channels (or multihop), that is, channels in which money can change hands on different occasions without the need for a transfer of funds to a particular wallet.

For the purpose of making a simpler description, but without taking away generality from the operation of the protocol, we will assume that M will use μ-coins of the same value in both channels (the protocol could easily be extended to the use of mu-coins of different value). The sub-protocol so that M can transfer the funds from the channel used with C to make payments through a new channel to another seller N is the following:

- M requests the smart contract that the μ-coins received go to another channel, to make payments at N, calling a function of the smart contract that manages the payment channels. This function can only be executed as an alternative to the channel liquidation function. To do this, N generates W_{0N} and M generates W'_{0M}, analogously to how W_{0M} and W_{0C} were generated in the opening process of the channel. $W_{0N} = H^n(W_{nN})$ and $W'_{0M} = H^n(W'_{nM})$. As we have already mentioned, in this case, the value of the μ-coins v is the same as in the transferred channel. On the other hand, the number of μ-coins in the channel can be equal to or less than the number of microcoins in the original channel $c' <= j$. Therefore, the value of n using is $n = c' * v$.
- M generates $\Gamma' = (W_{0N}, S_{IdN}, 2c', v, T'_{Exp}, \Delta'_{TD}, \Delta'_{TR}, W'_{0M})$.
- M can use the channel's μ-coins to make purchases in N, following the purchase sub-protocol described above. Finally, if desired, N would receive the money transfer through the channel's liquidation sub-protocol.

Refund

Once the channel has been settled, the μ-coins not used in payment operations will be returned to C. It has been decided that this reimbursement is not made automatically by the Smart Contract to allow C to decide if it wants to obtain the reimbursement or reconvert the channel for payment to a new seller.

The channel transfer procedure can also be used by C to create a new channel to exchange the unspent μ-coins for use in a new channel with another vendor M^*. This change can be made in the window between $(T_{Exp} + \Delta_{Td})$ and $(T_{Exp} + \Delta_{Td} + \Delta_{Tr})$.

These channel transfer operations allow reducing the costs associated with transfer operations on the blockchain.

15.3 Security properties analysis

In this section, the main properties of the channel management protocol for fair purchasing will be briefly described.

Anonymity

Users C and M, (and N if applicable) access the smart contract through their blockchain addresses.

However, the channel is associated with the knowledge of certain values. The user who knows the elements of the liquidation chain associated with the channel will be the user who can carry out the liquidation or transfer of the channel. Therefore, the user executes the protocol without the need for identification.

The fact that they are multi-hop channels, with the possibility of channel transfer without any movement of balances on the blockchain, allows a user to participate in purchase operations without the need for their account on the blockchain to have issued or received any transaction.

Equity

The purchase operation is considered an application of fair exchange of values. On the one hand, a payment is made and on the other, a service or product is provided.

In this protocol, the exchange is carried out without the intervention of any trusted third party. To carry out this fair exchange, an off-chain subprotocol is executed, where messages are exchanged directly between users C and M.

The procedure consists of three steps and could be interrupted without completing the operation. Possible breakpoints are as follows.

- After the first step of the exchange, where C provides the $\mu-$coin W_{iC}, M does not follow the protocol and does not provide the product or service.
  - In this case C will not send the proof element $W_{(i+1)C}$.
  - The purchase operation is considered not carried out. C does not receive the product or service and M cannot include the $\mu-$currency in the channel liquidation or transfer operation since it does not know the evidence element associated with the last $\mu-$currency .
  - In this first scenario, effectively M will not be able to collect the last $\mu-$coin but it could collect all the previous ones. Therefore, if the client wants to eliminate the risk of partially losing the last $\mu-$payment, by not receiving the corresponding service, he can adjust the value of the $\mu-$coin to each service and completely eliminate this conflict. However, it is foreseeable that M does not engage in this type of conduct, since by committing this $\mu-$fraud he would lose the following sales to the client and, therefore, would run the risk of losing more than what he has earned with this little fraud. In

any case, it is risk-controlled and regulated by the client, it is up to him to cancel or not this potential problem.

- After the second step of the exchange, C does not follow the protocol and does not provide the proof item associated with the $\mu-$coin, $W_{(i+1)C}$..
  - In this case the purchase is considered as completed.
  - C disposes of the acquired item while M cannot include the $\mu-$currency in the liquidation or transfer operations of the channel.
  - When making a new payment with the channel, C will use the $\mu-$coin $W_{(i+2)C}$. From this $\mu-$coin, M can derive the proof element of the previous buy operation $W_{(i+1)C}$.
  - In this scenario, a buyer can only leave a single $\mu-$coin per channel without validation, which falls within the acceptable risks in micropayment operations. On the other hand, the fraud associated with the creation of channels for the theft of individual $\mu-$coins is ruled out, taking into account that the creation of the channel implies execution costs that make it unfeasible to obtain a profit from the creation of a channel to proceed with the theft of a $\mu-$coin.

Transferability

The purchase protocol is based on the use of a payment channel established on the blockchain. This channel allows payment from a buyer C to a seller V. In a simple execution scenario, M will liquidate the channel after receiving payments with the $\mu-$coins associated with the channel. This liquidation will represent a transaction on the blockchain.

However, in order to increase the efficiency of the system, the protocol has been designed taking into account the possibility that the payment channels are *multihop*. In a *multihop* channel, $\mu-$coins can change hands several times without the need for each change of hands to represent a transaction on the blockchain.

The channel transfer operation allows multiple hops of the channel *multihop* to be performed. If you re-receive the $\mu-$coins the receiver, instead of making the channel liquidation, chooses to execute the transfer function, it can redirect the channel to make payments with it towards a new receiver. The channels can be prepared for a limited number of transfers since an unlimited transfer could lead to a loss of system efficiency and it must be taken into account that in the usual execution scenario the channels are not transferred more than a small number of times.

The protocol also contemplates the possibility that a buying user does not use all the $\mu-$coins in the channel before the expiration date of the channel. These funds can be redeemed in the buyer's account through a transaction operation programmed in the smart contract. This operation has not been automated, providing the buyer with the possibility of redirecting the channel to a new seller.

Impossibility of Falsification, Overspending and Double Spending

The protocol has been designed to prevent the falsification of $\mu-$coins as well as their reuse or the use of more $\mu-$coins than those established in the channel.

- All $\mu-$coins received as means of payment in the purchase protocol must belong to the associated channel. This check is performed by the operation $H^i(W_{iC}) == W_{0C}$, where W_{0C} must match the value stored within the Γ element of the channel.
- The reuse of a $\mu-$coin is avoided by checking $H^{i-j}(W_{iC}) == W_{jC}$, where i is the index of the $\mu-$current coin and j the index of the $\mu-$coin used in the last payment on the channel, so i must be greater than j.
- Overuse is avoided with two procedures. On the one hand, the money corresponding to the $\mu-$coins associated with the channel is deposited in the smart contract before starting the purchase operations. On the other hand, the number of $\mu-$coins associated with the channel is limited and the last $\mu-$coin in the channel is determined by the element $W_{(L-1)C}$. The seller will not accept any $\mu-$currency that requires validation on the hash chain with more than L operations.

Reduced Cost

A reduced cost per payment is a fundamental characteristic of $\mu-$payments since it should never exceed the benefit associated with that payment, much less the value of the amount transferred.

The protocol presented in this work considerably reduces the cost of a payment operation through cryptocurrencies by creating a payment channel. Although the number of payment operations that can be carried out is c operations per channel, the number of transfers on the blockchain is reduced to one channel, if transfer operations are not contemplated.

The more long-term the relationships between buyer and seller are, the longer the chains of $\mu-$coins associated with the channel can be, and the greater the efficiency of the system.

On the other hand, the design of multihop channels allows to reduce the number of real transfers on the blockchain, since a transferable channel does not require liquidation after use between the first two users.

15.4 Conclusions

In this chapter, we have presented a micropurchase scheme, that is, a system of equitable exchange between a micropayment and access to the purchased product or service. Micropayments are payments of very small amounts where the profit margin is low, so the system should not have associated costs that substantially reduce or completely eliminate that profit margin. To achieve this, the security and privacy mechanisms associated with micropayments must be designed accordingly.

In the case of cryptocurrencies, the solution to implement micropayments is to design layer two protocols [130] in which each payment operation is prevented from representing a transaction on the blockchain. A channel, once opened, allows the execution of off-chain operations that will not materialize on the blockchain until the moment of liquidation of the channel.

The chapter presents a channel management protocol using smart contracts. The system is based on the blockchain-free micropayment protocol presented in [121].

As in the [121] protocol, the payer builds a chain of elements where microcurrencies and proof elements are interspersed that will allow granting atomicity to the purchase operations.

The protocol consists of an initial service request phase followed by an on-chain operation in which the channel is created. The channel is not associated with a specific seller, but rather the seller will have an item that authenticates him or her as the channel's truthful liquidator. Next, the purchase operation is carried out as many times as necessary in an off-chain manner. Finally, the seller can liquidate the channel, which will proceed to the fund transfer transaction on the blockchain.

We have added channel transfer and refund operations to the basic protocol to increase system efficiency and avoid unnecessary transactions.

We also have provided an informal analysis of the properties of the system, including anonymity, fairness, transferability, impossibility of counterfeiting, overspending, and reuse of coins, as well as an assessment of the efficiency of the system.

CHAPTER 16

IDENTITY-RELATED ATTRIBUTE VERIFICATION PROTOCOL USING SBTS AND ZKPS

In the current context of growing concerns over privacy and the protection of personal data, the need for robust safeguards, particularly for highly sensitive Personally Identifiable Information (PII), has become increasingly paramount. This chapter presents an innovative approach to address these concerns through the utilization of Soulbound tokens (SBTs), Public-key cryptography, and Zero-Knowledge Proofs (ZKPs), within an identity verification framework. Through this integration, individuals gain control over their data, selectively disclosing it to trusted entities while upholding both privacy and security. Furthermore, this research contributes advancements to the verification of diverse attributes associated with the user's identity stored within an SBT and incorporating various comparator operators (equal, different, greater, less). Additionally, to prevent replay attacks, a timestamp attribute has been incorporated into the ZKP circuit. Relying on decentralized ledger technologies like blockchain, our protocol instills trust and transparency in identity management, fostering trust among stakeholders and mitigating fraud risks. Implemented using Solidity for smart contracts and Circom for ZKPs, the protocol undergoes a comprehensive analysis encompassing performance metrics and security properties. The obtained results reveal that the costs associated with the protocol are economically viable. We also demonstrate that the inherent properties of the employed technologies afford critical security assurances. Additionally, we outline several practical applications and use cases of these technologies, such as digital voting, online banking, and e-commerce. The convergence of public-key cryptography, SBTs and ZKPs presents a transformative paradigm for Self-Sovereign Identity (SSI) and data privacy. By endowing individuals with greater autonomy over their personal data and enabling secure, decentralized data sharing, these technologies lay the groundwork for a trust-centric digital society.

16.1 Contribution

In this chapter, we delve into a pioneering approach to decentralized and confidential digital identity management through the integration of Zero-Knowledge Proofs (ZKPs) and Soulbound tokens (SBTs). Our work introduces a novel protocol, termed zkSBT, which combines the encryption of SBTs using the public key of the identity holder with ZKPs for the verification of identity attributes without compromising personal information. This integration not only advances the privacy and security features in digital identity verification but also marks a significant leap from existing methodologies.

To the best of our knowledge, the present work constitutes the first proposal that jointly considers the use of SBTs, ZKP, and the private/public key of the identity holder in a comprehensive approach to develop an Identity-Related Attribute Verification Protocol. This integration represents a key strength of our proposal, where these three elements converge to represent attributes related to personal identity, facilitating secure and private data sharing in a decentralized and distributed manner. The features of our proposal align with the principles of SSI for several reasons:

- **Immutability and Non-Transferability:** SBTs within an SSI framework represent immutable credentials tied to an individual's identity, ensuring that the credentials or attributes they represent cannot be transferred or sold. This non-transferability emphasizes personal control over identity and credentials.
- **Blockchain Verification:** SBTs can be verified on a blockchain, providing a transparent and tamper-proof method for confirming the authenticity of an individual's credentials, thereby enhancing the trustworthiness of the SSI system.
- **Enhanced Privacy through ZKPs:** By integrating ZKPs with SBTs, individuals can prove certain attributes or credentials without revealing the underlying data, which strengthens privacy within an SSI framework.
- **Advanced Verification Mechanisms:** Our protocol introduces a novel advanced verification mechanism that allows the validation of specific identity-related data against predetermined criteria, providing an effective solution for attribute verification in identity contexts.
- **Encryption for Enhanced Privacy:** Encrypting the identity holder's data stored in the SBT with their public key further enhances privacy.

As a result, the proposed zkSBTs complement SSI by providing a robust method for representing and verifying personal credentials in a non-transferable, privacy-preserving manner, thus enhancing the overall trust and utility of digital identity systems.

A cornerstone of our contribution is the utilization of the Non-Transferable Tokens (NTTs), as standardized by EIP-4671, which enhances the interoperability and security of the SBTs across various blockchain ecosystems. Distinctively, EIP-4671 facilitates the revocation of tokens without their deletion, fostering a more secure and coherent digital identity infrastructure. Furthermore, the protocol leverages the immutable and secure nature of blockchain technology, thus reinforcing the trust and integrity in Self-Sovereign Identity (SSI) systems.

To accommodate a variable amount of data, our protocol introduces a dynamic data storage solution employing a Merkle tree structure. This feature is complemented by an advanced data verification mechanism that supports various comparator operators, including *equal, different, greater than, greater or equal than, less than* and *less or equal than*, extending the protocol's utility to a wider array of applications. Furthermore, to mitigate replay attacks, a timestamp attribute has been integrated into the ZKP circuit. Our implementation and testing on the Polygon PoS Chain demonstrate the protocol's viability across any EVM-compatible network, showcasing its broad applicability and ease of integration into existing blockchain infrastructures.

Additionally, the use of tokens offers several significant advantages, particularly in terms of traceability and auditability. Tokens allow for the tracking and auditing of their creation process, including when they are minted, who created them, and when they were accepted. Furthermore, as an Ethereum Improvement Proposal (EIP) standard, similar to fungible and non-fungible tokens, they benefit from the advantages of standardized smart contracts. These contracts share interfaces (functions) with other smart contracts of the same type, enabling their generic use across various applications, such as listing on web services.

Although revocability is an important consideration, it falls outside the scope of the present proposal and is currently addressed as future work. At present, there is no established protocol for an authority to remove a SBT from an account. Nevertheless, revocation can be managed by maintaining a database of revoked SBTs or by incorporating a 'revoked' field that only the issuing authority can modify. The existing protocol does support the inclusion of an expiration date as an additional field within the SBT, which would provide temporal validity to the identity information of the identity holder.

The employment of *Circom* for circuit compilation and *snarkjs* for the generation and verification of zkSNARK proofs exemplifies the efficiency and user-friendliness of our approach. Opting for the Poseidon [55] hashing algorithm underscores our commitment to leveraging cutting-edge technology for optimal security and privacy outcomes. Our comprehensive analysis of the protocol's security properties, alongside the exploration of its versatility through various use cases, underscores the robustness and practical applicability of our solution in enhancing user privacy and security.

Moreover, the performance evaluation of the smart contracts associated with our protocol ensures that our solution is not just secure and private but also scalable and cost-effective for real-world applications. Through this integrated approach, combining the latest technological advancements with rigorous testing and detailed analysis, our contribution sets a new standard in the field of blockchain-based digital identity management, particularly within the context of e-commerce. This work paves the way for future innovations in secure and user-centric digital identity solutions, significantly enhancing the privacy and security landscape of digital identities in SSI systems.

16.2 Protocol design

The Identity-related attribute verification protocol proposed in this chapter comprises three essential actors: the *Identity holder*, the *Authority* or *Issuer*, and the *Verifier* (See Figure 16.1).

The ***Identity holder*** is the person who owns personal identity information. He has sovereignty over who can access its identity-related information and he can choose which aspects to share with each external entity. In our protocol, the *Identity holder* stores their information in an SBT issued by the *Authority*. Additionally, he generates a ZKP to prove its identity or identity-related attributes to a third party without disclosing any extra personal information.

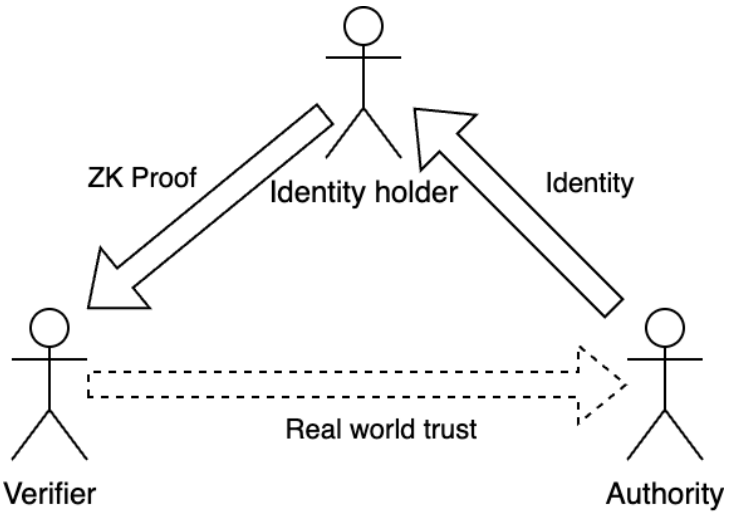


Figure 16.1: Participants of the zkSBT protocol

The ***Authority*** **(or *Issuer*)** is the entity responsible for certifying the identity information of the *Identity holder*. Authorities can encompass various organizations or entities with a legitimate interest in confirming individual identities, such as governments, banks, or employers. The *Authority* generates digital credentials attesting to specific attributes of the *Identity holder*, such as name, date of birth, or educational qualifications. In our protocol, these credentials are encrypted using the *Identity holder*'s public key and then stored on a blockchain within an SBT that is transferred to the *Identity holder*.

The ***Verifier*** user is the entity seeking access to some data regarding the identity information of the *Identity holder*. Verifiers can be organizations or entities requiring identity verification or confirmation of specific identity-related attributes, including financial institutions, government agencies, or online service providers, among others. The *Verifier* initiates a request for access to specific attributes within the *Identity holder*'s information and validates the genuineness of digital credentials issued by the *Authority* through cryptographic proofs. The *Verifier* places trust in the credibility of the digital credentials issued by the *Authority*, given their storage on a decentralized ledger, ensuring they remain unaltered and immune to forgery.

Within the SSI model, individuals have absolute control over their personal identity information, allowing them the discretion to share it with verifiers as required. This empowers individuals with enhanced privacy, security, and authority over their identity details, simultaneously diminishing dependence on centralized identity management systems. The incorporation of cryptographic methods, decentralized ledgers, and digital credentials guarantees the security, reliability, and resistance to tampering of

Table 16.1: Notation of the decentralized and confidential digital identity using ZKP protocol

Name	Description
I	*Identity holder* user
A	*Authority* or *Issuer* user
V	*Verifier* user
$H()$	Secure hash algorithm
p	Private key of *Identity holder*
P	Public key of *Identity holder*
$d = d_1, d_2, ...d_n$	Identity data
$D = D_1, D_2, ...D_n$	Encrypted identity data
a_I	Address of *Identity holder* user
$H(d)$	Merkle tree root of identity data d
C	Circuit
$lambda$	Random number to generate proving and verification keys
pk	Proving key of the ZKP
vk	Verification key of the ZKP
i	Index in the data array d of the value that we want to verify
o	Operator to compare the value d_i with the threshold t (==, !=, >, >=, <, <=)
t	Threshold that the user needs to meet
ts	Timestamp indicating the specific moment in time when the verification request is made, used to prevent replay attacks
prf	Proof generated by the *Identity holder*

the SSI model.

We employ the notation detailed in Table 16.1 to describe the protocol. The fundamental concept behind the proposed protocol is to encrypt identity-related data with the *Identity holder*'s public key, encapsulating it within an SBT. Subsequently, the *Identity holder* can provide proof of the data to a verifier through the utilization of a ZKP without disclosing the data content.

Initially, the system provider needs to create the system setup. This involves generating a Zero-Knowledge Soulbound Token (zkSBT) smart contract, wherein any user can mint, because any account can act as an *Authority A*. Allowing any user to mint is not a problem, as each *Verifier V* will decide which authorities to trust, and thus which zkSBTs will support for validation. The zkSBT smart contract will be an SBT (an ERC-721 compatible token that cannot be transferred) that will contain all encrypted data of the *Identiy holders* encrypted with their public key P.

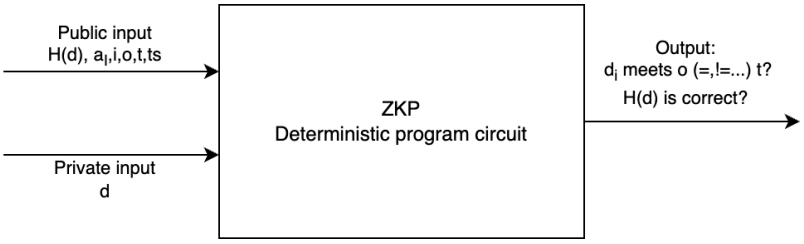


Figure 16.2: ZKP deterministic program circuit

Before executing the protocol, the system provider also needs to establish a ZKP system for validating a specific claim. The generation of this ZKP system involves the following steps:

1. Develop a circuit C (see Figure 16.2) designed to verify that the data fulfills specific criteria (having a value d_i that meets an operation o through a defined threshold t, for example, that the *Identity holder* has a credit score greater or equal than 10) and that the Merkle tree root of this data $H(d)$ corresponds to the Merkle tree root stored in the zkSBT.

2. The circuit C will need a private input, the data d, along with some public inputs: the Merkle tree root of the data $H(d)$, the index i of the data in the data array d (to know which value we want to check from the data), the threshold t, and the operator o that we want to use to compare the value d_i with the threshold t. Possible operators are: ==, !=, >, >=, <, <=. For example, with this information, we will be able to check if a user is older than 18 years using the operator >= (i.e. $o \leftarrow$ >=), and $t = 18$ for a date of birth attribute. Another example could be the case of a user has a credit score greater or equal to a certain threshold using: $o \leftarrow$ >=, and t set to the desired credit score threshold.

3. We also will incorporate another public value to the circuit, the address of the *Identity holder* user, a_I, to avoid collisions in the Merkle tree root hashes when we have few values stored in that tree of data. To avoid these collisions, we need to add this address a_I to the Merkle tree data as its first element. Incorporating the user's address into the Merkle tree is a strategy to enhance the uniqueness of each leaf node, thereby reducing the chances of hash collisions, especially when the number of values stored in the tree is relatively small.

4. Additionally, we will add a ts (timestamp) public value to the circuit. The timestamp ts indicates the specific moment in time when the verification request is made and is used to prevent replay attacks. By incorporating the timestamp into the ZKP circuit, we ensure that each proof is unique to the specific verification request and time, thereby preventing malicious actors from reusing previously submitted proofs. This enhancement is essential for maintaining the integrity and security of the verification process.

5. Create a pair of cryptographic keys for the circuit, consisting of a proving key pk and a verification key vk. These keys are derived from a secret parameter $lambda$ and the circuit C. They serve as public parameters and require generation only once for a specific circuit C.

6. Using tools like *snarkjs*, it becomes feasible to automatically generate a smart contract for validating the proof. This process entails integrating public inputs, the verification proof *prf*, and the verification key vk. Through this utility, we can establish a `verifier.sol` smart contract designed to verify the validity of the proof. This `verifier.sol` smart contract can be invoked from the zkSBT smart contract, adding the capability to confirm the authenticity of the proof.

7. In the smart contract's side we will also need to include a check to verify that the submitted zkSBT belongs to the *Identity holder* user and that this zkSBT shares the same hash as the data $H(d)$ submitted by the *Identity holder* user to the ZKP verification function.

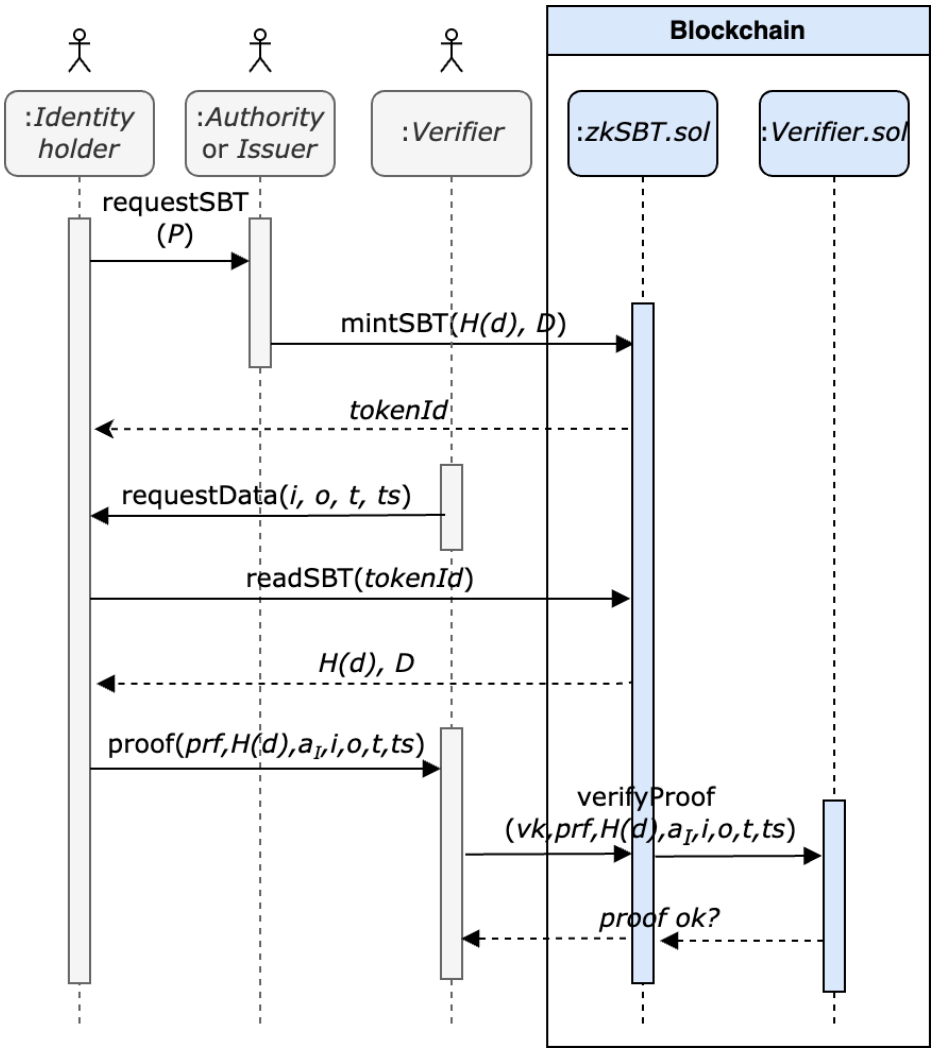


Figure 16.3: ZKP Soulbound token sequence diagram, with roles, smart contracts, functions and their parameters

The steps of the protocol are detailed in Figure 16.3, where we depict the general sequence diagram of the protocol. In this diagram, we can see the three different actors (*Identity holder*, *Authority*, and *Verifier*), that will be able to interact with the system with a web interface, the smart contracts deployed on the blockchain (`ZKSBT.sol` and `verifier.sol`), and all the messages (methods or functions) invoked by these actors with their respective parameters. The steps outlined in the diagram are elaborated in the following points:

1. The *Identity holder I* requests a zkSBT from the *Authority A*. To facilitate this, I must send their public key P to the *Authority A*. The *Identity holder* can either generate a dedicated public-private key pair for this transaction or use the public key linked to their blockchain address. While the public key P can be shared with anyone, the private key p must remain confidential.

2. The *Authority A* uses the public key P to encrypt the identity data d. This process can be executed through an asymmetric encryption algorithm such as RSA or Elliptic Curve Cryptography (ECC). However, it is important to note that encryption with asymmetric cryptosystems is typically not recommended for large amounts of data, and it is more suitable for encrypting small blocks of data, such as a secret key. We specifically employ asymmetric encryption for small 32-byte blocks, emphasizing its appropriateness for this particular context.

   The data d represents a claim regarding the user seeking the zkSBT and is endorsed by the *Authority*, given that verifiers place trust in A. The *Authority* also needs to calculate the Merkle tree root of the data d using the ZK-friendly hashing algorithm *Poseidon* [55]. In situations where the encrypted information is excessively large, an alternative approach is to upload this encrypted data D to services like IPFS and store in the zkSBT only the corresponding URL pointing to this file, together with the Merkle tree root of the data $H(d)$. These calculations are shown in Subprotocol 27.

   To mint the SBT the *Authority* must send both the encrypted data D and the Merkle tree root of the data $H(d)$.

Subprotocol 27: Encrypt the data

1. A encrypts: $D = encrypt_P(d)$
2. A calculates: $H(d) = merkleTreeRoot(d)$

3. After minting the zkSBT, the corresponding $tokenId$, which uniquely identifies the newly created token, is then forwarded to the *Identity holder*.

4. When a verifier seeks specific data from the *Identity holder*, they request the index i of the value that we need to check inside the data d, a particular threshold t, and an operator o that the piece of data d_i must satisfy, applied to threshold t. He will also need to provide the timestamp ts of the request. For instance, the *Verifier* might ask for a credit score d_i greater or equal (o) than a specified threshold t. The communication exchange between I and V takes place over a secure channel that provides end-to-end authenticity, integrity and confidentiality (e.g. an SSL/TLS channel[1]).

5. The *Identity holder* reads the data stored in their zkSBT, uniquely identified by a $tokenId$.

6. The retrieved information encompasses both the Merkle tree root of the data ($H(d)$) and the Encrypted identity data (D).

[1] Guide to TLS Standards Compliance: https://www.ssl.com/guide/tls-standards-compliance/

7. After reading the stored data in their zkSBT, the *Identity holder* proceeds to decrypt it using their private key p. This step results in the *Identity holder* obtaining their own decrypted identity data d. This decryption process is performed off-chain, within the identity holder's own computer, to ensure that sensitive information is not revealed or exposed during the process.

   Subsequently, the *Identity holder* produces the proof *prf* by using the circuit C, the cryptographic proving key pk, and the inputs for the computation (including the private data d and the public values: the Merkle tree root of the data $H(d)$, the *Identity holder* address a_I, the index of the value i, the operator o, and the threshold t, and the timestamp ts). This process is also performed off-chain, within the identity holder's own computer, to avoid revealing any private information. Through this process, the *Identity holder* generates a proof *prf* demonstrating the accurate execution of the computation without disclosing any private inputs or intermediate computations and sends it to the *Verifier*.

   All these steps needed by the *Identity holder* to generate *prf* are shown in Subprotocol 28.

Subprotocol 28: Generate the proof

1. I decrypts: $d = decrypt_p(D)$
2. I calculates: $H(d) = merkleTreeRoot(d)$
3. I generates: $prf = C(pk, d, H(d), a_I, i, o, t, ts)$

8. Upon receiving the proof *prf*, the *Verifier* calls the `verifyProof()` function of the zkSBT, which invokes the previously described `verifier.sol` smart contract. The verification process involves confirming that the provided zkSBT belongs to I, that the zkSBT is issued by A, that the Merkle tree root of the data sent by the *Identity holder* to the *Verifier* aligns with the one stored in the zkSBT ($H(d)$), and the data complies with the criteria outlined in the circuit C, ensuring that the value d_i satisfies a specified operator o through a threshold t and that the sent data d matches the Merkle tree root of the data $H(d)$, and that the timestamp ts is valid. These checks are shown in Subprotocol 29.

Subprotocol 29: Verify the proof

1. V checks: **if** $((ownerOf(tokenId) == I)$ &&
   $(minterOf(tokenId) == A)$ &&
   $(getRoot(tokenId) == H(d))$ &&
   $(verifyProof(C, vk, H(d), a_I, i, o, t, ts)))$ **then**
   $proofIsValid$
   end if

9. If the verification smart contract provides a positive confirmation, the verifying user can be assured that the user's private data d_i satisfies the threshold t without having access to the actual private data d.

Considering all aspects, we accomplish the following objectives:

- The information stored on-chain within the zkSBT remains confidential, and only *I* has the capability to decrypt it using their private key p.
- The verifying user can be confident that the identity data is originated from the *Authority A*.
- The verifying user can ensure that the *Identity holder* has not manipulated the data by verifying the Merkle tree root of the on-chain stored data.
- The verifying user can specify any requirement or threshold that users must fulfill.
- The verifying user can be sure that the received information is not reused thanks to the introduction of the *timestamp*.
- Although the proof *prf* is generated by the *Identity holder* off-chain, no private data is disclosed.
- Using ZKPs, the *Identity holder* can demonstrate compliance with a specific threshold t without disclosing any of their personal information or data.

16.3 Implementation

We have written some smart contracts in Solidity language to implement, test, and check the performance of the zkSBT, following the protocol presented in section 16.2. The code for this smart contract and their corresponding tests can be accessed in the *zksbt*[2] GitHub repository, which is maintained by the Security and e-Commerce (SECOM) Research Group from the University of the Balearic Islands. For evaluating performance and gas costs, we employed Polygon PoS, an Ethereum Virtual Machine (EVM) compatible blockchain. EVM-compatible blockchains use programs known as smart contracts and a distributed Turing Complete machine to record changes in the system state. These EVM blockchains leverage their native currency, such as Ethers (ETHs) in Ethereum or Matic in Polygon PoS, to measure and restrict the costs of resources used in code execution. In this Section, we will see the development procedure and the description of the necessary Smart Contracts.

To test the complete workflow of the zkSBT protocol, we also have implemented a prototype user interface. The code for this interface is available on GitHub[3]. The application is deployed on Vercel and can be accessed at zkSBT Vercel App[4]. With this UI we also aimed to evaluate the protocol's performance, security, and usability in a real-world scenario.

To implement the described protocol, we also need to write some circuits to generate the ZKPs. We will use *Circom*, a compiler written in Rust for compiling circuits written in the *Circom* language. It is part of the *Circom* ecosystem, which also includes a library of templates called *circomlib*, and the *snarkjs* tools for testing and working with *Circom* circuits.

[2] https://github.com/secomuib/zksbt
[3] https://github.com/secomuib/zksbt-ui
[4] https://zksbt.vercel.app/

Zero-knowledge proof circuit

As we have seen in the protocol description (section 16.2), we need to create the system setup, that involves the generation of a ZKP circuit C that will let *Identity holder I* generate a proof *prf* that he owns an SBT that includes a piece of data d_i that meets an operation o through a defined threshold t.

To do this, we will use *Circom*, a language designed for creating ZKP circuits. It is specifically tailored for the development of circuits used in Zero-Knowledge Succinct Non-Interactive Argument of Knowledges (zkSNARKs), a type of ZKP.

To compile and generate the *Circom* circuits, we will set up a new *Node.js* project with the `npm init` instruction from the command line. Then, we will install some packages needed for *Circom* inside this project using the command `npm install circomlib circomlibjs snarkjs`.

Before writing the main circuit, we will write two *Circom* templates, `Compare` and `verifyZKSBT`. A template is a predefined circuit structure that can be reused for similar tasks. Templates provide a way to modularize and reuse common components across different circuits, making the development process more efficient.

The first template is for comparing two inputs based on the given operator `op` (see Listing 33). This template is taken from the GPL licensed repository *zkcertree*[5].

The `Compare` template takes two inputs `a` and `b`, and an operator `op`. The operator is a 3-bit bitmap, representing different comparison operations. The circuit then returns an output `out` which can be 1 if the condition is satisfied or 0 otherwise.

Here is a step-by-step explanation of the circuit:

- The circuit starts by including two external *Circom* libraries, `mux3.circom` and `comparators.circom`. These libraries provide the components used in the circuit, such as the multiplexer (`mux`) and the comparison components (`LessThan`, `IsEqual`, `GreaterThan`, etc.).
- The `Compare` template is declared with four signals: `a`, `b`, `op`, and `out`. `a` and `b` are the inputs to be compared, `op` is the operator representing the comparison operation, and `out` is the output of the comparison.
- The `validOp` component is a `LessThan` comparator with a limit of 252 bits. It checks if the operator `op` is less than 6, which is the number of supported operations. If the operator is not supported, the `assert` statement will fail.
- The `eq`, `gt`, `gte`, `lt`, and `lte` components are comparison components that compare `a` and `b` using the corresponding operations: equal to, greater than, greater than or equal to, less than, and less than or equal to.
- The `mux` component is a 3-bit multiplexer. It uses the `n2b` component to convert the operator `op` into a 3-bit binary number, which is used as the selector for the multiplexer.
- The selectors for the multiplexer are set to the outputs of the comparison components. This means that the multiplexer will select the output of the comparison operation that corresponds to the operator `op`.

[5] https://github.com/r0qs/zkcertree

```
pragma circom 2.0.4;

include "../../node_modules/circomlib/circuits/mux3.circom";
include "../../node_modules/circomlib/circuits/comparators.circom";

// Ret: 1 if condition is satisfied. 0 otherwise
template Compare() {
    signal input a;
    signal input b;
    signal input op;
    signal output out;

    component validOp = LessThan(252);
    validOp.in[0] <== op;
    validOp.in[1] <== 6;
    // Restrict to supported operators
    assert(validOp.out);

    component eq = IsEqual();
    eq.in[0] <== a;
    eq.in[1] <== b;
    component gt = GreaterThan(252);
    gt.in[0] <== a;
    gt.in[1] <== b;
    component gte = GreaterEqThan(252);
    gte.in[0] <== a;
    gte.in[1] <== b;
    component lt = LessThan(252);
    lt.in[0] <== a;
    lt.in[1] <== b;
    component lte = LessEqThan(252);
    lte.in[0] <== a;
    lte.in[1] <== b;

    component mux = Mux3();
    component n2b = Num2Bits(3);
    n2b.in <== op;
    mux.s[0] <== n2b.out[0];
    mux.s[1] <== n2b.out[1];
    mux.s[2] <== n2b.out[2];

    mux.c[0] <== eq.out;      // 000: ==
    mux.c[1] <== 1 - eq.out; // 001: !=
    mux.c[2] <== gt.out;      // 010: >
    mux.c[3] <== gte.out;     // 011: >=
    mux.c[4] <== lt.out;      // 100: <
    mux.c[5] <== lte.out;     // 101: <=
    mux.c[6] <== 0;
    mux.c[7] <== 0;

    mux.out ==> out;
}
```

Listing 33: Code of the circuit compare.circom

```
pragma circom 2.0.0;
include "../../node_modules/circomlib/circuits/poseidon.circom";
include "./compare.circom";

// length = length of the data array
template verifyZKSBT(length) {
    // public
    signal input index;
    signal input root;
    signal input owner;
    signal input threshold;
    signal input operator;
    signal input timestamp;

    // private
    signal input value;
    signal input data[length];

    signal output out;

    assert(owner == data[0]);
    assert(value == data[index]);
    assert(timestamp > 0);

    component merkleTree = Poseidon(length);
    for (var i = 0; i < length; i++) {
        merkleTree.inputs[i] <== data[i];
    }
    root === merkleTree.out;

    component cmp = Compare();
    cmp.a <== value;
    cmp.b <== threshold;
    cmp.op <== operator;
    cmp.out === 1;

    // output
    out <== cmp.out;
}
```

Listing 34: Code of the circuit verifyZKSBT.circom

- The output of the multiplexer is the output of the `Compare` circuit. It will be 1 if the condition is satisfied (i.e., if the selected comparison operation returns 1), and 0 otherwise.

Another circuit that we need to write is a template for generating a zkSBT proof (see Listing 34). It's used to verify that a certain condition is met without revealing any information about the data inside the zkSBT being verified.

The circuit is defined as a template `verifyZKSBT` which takes a single parameter `length`, which represents the length of the data array.

There are several public inputs:

- `index`: The index i of the value in the data array.

```
pragma circom 2.0.0;

include "verifyZKSBT.circom";

component main {public [
    index, root, owner, threshold, operator
    ]} = verifyZKSBT(4);
```

Listing 35: Code of the circuit verify4.circom

- `root`: The root of the Merkle tree $H(d)$.
- `owner`: The address of the owner of the SBT (*Identity holder* address) a_I.
- `threshold`: The threshold t to compare with the value.
- `operator`: The operator o to compare the value d_i with the threshold t.
- `timestamp`: Timestamp ts of the proof.

The circuit also has a private input `value` (d_i) and an array of private inputs `data` (d) of length `length`. It also has one output `out`, which is a boolean value indicating whether the proof is valid or not.

There are included two templates: `Poseidon` and `Compare`, previously defined. The circuit creates a `merkleTree` component that uses the `Poseidon` template to create a Merkle tree and calculate the root using the Poseidon hash algorithm. The Merkle tree is a binary tree of hashes, where each non-leaf node is the hash of its children. The root of the Merkle tree is a hash of all the data in the tree. The `merkleTree` component takes the `data` array as input and produces the root of the Merkle tree as output. The circuit checks that the root of the Merkle tree is equal to the `root` ($H(d)$) public input.

Then, the circuit creates the `cmp` component that uses the `Compare` template, already defined in this section. It takes three inputs: `a`, `b`, and `op`. The `a` and `b` inputs are the values to be compared, the value d_i and the threshold t, and the `op` input is the operator o to be used for the comparison. The `cmp` component produces a boolean output indicating whether the comparison is true or false, that is, that the value d_i meets certain criteria over the threshold t, applying the operator o. The circuit checks that this comparison is true.

The circuit also includes several assertions. The first assertion checks that the `owner` (a_I) is equal to the owner in the data (we will always set it as the first element of the data array). The second assertion checks that the `value` (d_i) is equal to the data at the `index` (i) position. The third assertion checks that the `timestamp` (ts) is valid. In addition, the circuit sets the `out` output to be equal to the output of the `cmp` component. If the comparison is true, the `out` output will be 1; otherwise, it will be 0.

Finally, we will define the `verify4` circuit, which will call the `verifyZKSBT` template defined previously. This circuit is used to verify a zkSBT data condition, without revealing any information about the data being verified. In this case, we define a data array of size 4 to use in our example demonstration. We can see the `verify4` circuit in Listing 35.

The circuit has several public inputs:

- `index`: The index i of the value in the data array.
- `root`: The root of the Merkle tree $H(d)$.
- `owner`: The address of the owner of the SBT (*Identity holder* address) a_I.
- `threshold`: The threshold t to compare with the value.
- `operator`: The operator o to compare the value d_i with the threshold t.
- `timestamp`: Timestamp ts of the proof

The circuit also has a private input `value` (d_i) and an array of private inputs `data` (d) of length 4.

The circuit has one output `out`, which is a boolean value indicating whether the proof is valid or not.

Compile the zero-knowledge proof circuit and generate the proving and verification keys

As a prerequisite, to compile the ZKP circuit, we need to have installed the *Circom* ecosystem (*Circom* and *snarkjs*) following its installation instructions[6].

We can compile the `verify4` circuit described before, using the following command:

```
$ circom verify4.circom --r1cs --wasm
```

- `-r1cs` will generate the file `verify4.r1cs` that contains the R1CS constraint system[7] of the circuit in binary format.
- `-wasm`: will generate the directory `verify4_js` that contains the `Wasm` code (`verify4.wasm`) and other files needed to generate the witness[8].

Download the trusted setup, specifically the Powers of Tau file which is a community-generated trust establishment (`powersOfTau28_hez_final_11.ptau`[9]). A trusted setup refers to an algorithm designed to derive a protocol's public parameters using confidential information to safeguard the security of the protocol. The Powers of Tau file, in this context, is a crucial component of the trusted setup process.

Generate the proving and verification keys, starting from `verify4.r1cs` (containing the circuit's description and constraints) and `powersOfTau28_hez_final_11.ptau` (the trusted setup). The resulting file from this process is `verify4.zkey`, representing a zero-knowledge key that includes both the proving and verification keys (pk and vk) associated with the circuit.

```
$ snarkjs groth16 setup \
    verify4.r1cs \
    powersOfTau28_hez_final_11.ptau \
    verify4.zkey
```

[6] https://docs.circom.io/getting-started/installation/
[7] https://docs.circom.io/background/background#rank-1-constraint-system
[8] https://docs.circom.io/background/background#witness
[9] https://hermez.s3-eu-west-1.amazonaws.com/powersOfTau28_hez_final_11.ptau

```
interface IERC4671 {
  event Minted(address owner, uint256 tokenId);

  event Revoked(address owner, uint256 tokenId);

  function balanceOf(address owner) external view returns (uint256);

  function ownerOf(uint256 tokenId) external view returns (address);

  function isValid(uint256 tokenId) external view returns (bool);

  function hasValid(address owner) external view returns (bool);
}
```

Listing 36: Interface of an ERC-4671 token

Verifier ZKP smart contract

At this point, we can generate a Solidity *verifier* smart contract using the *snarkjs* tool, which facilitates the creation of a Solidity smart contract for proof validation. This process starts with the `verify4.zkey` file, and the result is the `verifier.sol` file produced by the program.

```
$ snarkjs zkey export solidityverifier \
    verify4.zkey verifier.sol
```

This `verifier.sol` contract contains a function called `verifyProof()` that can be used to verify the proofs on an EVM-compatible blockchain. The verification function takes the proof and the public inputs to the computation, and returns whether the proof is valid.

Soulbound token smart contract

Instead of implementing our own version of an SBT, we will use the EIP-4671 standard. The EIP-4671 token can be used as an SBT because it implements the Non-Transferable Token (NTT) standard, which is designed to represent inherently personal possessions, such as university diplomas, online training certificates, government-issued documents, and so on. These tokens are meant to be non-tradable or "soulbound", meaning they can't be transferred, and they include all necessary functions and properties required to have an SBT implementation. It is important to note that this standard doesn't inherit from the ERC-721 standard, which has transfer-related functions that we don't need.

In Listing 36 we can see the interface of an ERC-4671 token. Its implementation is public in a public repository [10].

The IERC4671 interface includes the following events and functions:

- `Minted` event: This event is emitted when a token is minted for a specific owner. It includes the `owner` address and the `tokenId` of the minted token.
- `Revoked` event: This event is emitted when a token is revoked. It includes the `owner` address and the `tokenId` of the revoked token.

[10] https://github.com/ethereum/ERCs/blob/master/assets/erc-4671/ERC4671.sol

- `balanceOf()`: This function returns the number of tokens owned by a specific address. It takes an `owner` address as an input and returns a `uint256` representing the number of tokens owned.

- `ownerOf()`: This function returns the owner of a specific token. It takes a `tokenId` as an input and returns the `address` of the owner.

- `isValid()`: This function checks if a token is valid (i.e., it hasn't been revoked). It takes a `tokenId` as an input and returns a `bool` indicating whether the token is valid.

- `hasValid()`: This function checks if an address owns a valid token. It takes an `owner` address as an input and returns a `bool` indicating whether the owner has a valid token.

Zero-knowledge Soulbound token smart contract

To implement the zkSBT smart contract, we will inherit from the ERC-4671 smart contract implementation, to have all the necessary SBT-related functions and properties, and then we will add all ZKP related code. This will include all needed functions and properties to mint a new zkSBT, and to write and get the information stored inside this kind of token.

In Listing 37 we can see the basic structure of the zkSBT smart contract. This code defines a contract named `ZKSBT.sol` that extends the `ERC4671` contract. The `ERC4671` contract, as we have explained before, is a standard for creating Non-Transferable Tokens (NTTs), also known as Soulbound tokens (SBTs), on the Ethereum blockchain.

The `ZKSBT.sol` contract includes the following elements:

- Interface `IVerifier`: This interface declares a function `verifyProof` that takes in two arrays of unsigned integers and an array of unsigned integers, and returns a boolean value. These parameters are used to verify a proof using the `verifier.sol` smart contract sending the cryptographic proof and the public parameters.

- Internal variable `_verifier`: This variable is of type `IVerifier` and is used to store the `verifier.sol` smart contract that will be used to verify proofs.

- Struct `SBTData`: This struct includes two fields: `root` and `encryptedData`. The `root` field is a byte array, and represents the Merkle tree root of the data associated with a zkSBT, and the `encryptedData` field is an array of byte arrays, that represents all the encrypted data stored in each token.

- Mapping `sbtData`: This mapping maps a `uint256` token ID to an `SBTData` struct. This mapping is used to store the data for each SBT.

- `constructor` function: The constructor takes in three parameters: a string for the name of the token, a string for the symbol of the token, and an `IVerifier` contract. It initializes the `ERC4671` contract with the provided `name` and `symbol`, and sets the `_verifier` variable to the provided `verifier.sol` contract.

```
pragma solidity ^0.8.18;

import "./eip-4671/ERC4671.sol";

interface IVerifier {
    function verifyProof(
        uint[2] memory a,
        uint[2][2] memory b,
        uint[2] memory c,
        uint[7] memory input
    ) external view returns (bool);
}

contract ZKSBT is ERC4671 {
    IVerifier internal _verifier;

    struct SBTData {
        bytes root;
        bytes[] encryptedData;
    }

    mapping(uint256 => SBTData) public sbtData;

    constructor(
        string memory name,
        string memory symbol,
        IVerifier verifier
    ) ERC4671(name, symbol) {
        _verifier = verifier;
    }
}
```

Listing 37: General structure of the ZKSBT smart contract

We also need to define a `mint` function, that lets any authority create a new zkSBT, send it to any *Identity holder I*, and store the encrypted data D. It lets mint to any account because any account can act as an authority, because of the principles of self-sovereignty and the decentralized nature of the system.

The `mint` function is listed in Listing 38, and it includes the following parameters:

- `to`: This is the address of the *Identity holder I* to which the new token will be minted.
- `root`: This is a byte array that likely represents the root of the Merkle tree ($H(d)$) of the data that we will associate with this new zkSBT.
- `encryptedData`: This is an array of byte arrays that represents encrypted data D associated with the token.

The function calls the `_mint` private function of the parent `ERC4671` contract, to mint a new token and assign it to the address of the *Identity holder*. The function then creates a new `SBTData` struct with the provided `root` and `encryptedData`, and stores

```
function mint(
    address to,
    bytes calldata root,
    bytes[] calldata encryptedData
) external payable virtual returns (uint256) {
    uint256 tokenId = _mint(to);

    sbtData[tokenId] = SBTData({
        root: root,
        encryptedData: encryptedData
    });

    return tokenId;
}
```

Listing 38: mint() function

it in the `sbtData` mapping with the `tokenId` as the key. Finally, the function returns the `tokenId` of the newly minted token.

Finally, we also need to create a function called `verifyProof`, which is depicted in Listing 39. This function is used to verify the ZKP, sending the `tokenId` of the zkSBT, and the cryptographic proof and the public values of the ZKP.

The `verifyProof` function first retrieves the owner of the token and checks if the owner is the same as the address that is stored in the public values of the proof. The function then retrieves the root of the Merkle tree associated with the token and checks if it matches the root stored in the proof. Then, the function formats the proof and the public values in the format expected by the `_verifier` smart contract and calls the `verifyProof` function of that *verifier* contract to verify the proof. Finally, the function returns `true` if the proof is verified successfully.

Another two functions that we will need are the `getRoot` and `getEncryptedData` to read the stored sbtData from a specific `tokenId`. These two functions are detailed in Listing 40. These functions simply return the root of the Merkle Tree's data without encryption $H(d)$ and encrypted data D, given a specific `tokenId`.

Once the `ZKSBT.sol` and `verifier.sol` smart contracts are completely defined, we can deploy them to any EVM-compatible network, like Ethereum or Polygon PoS using tools like Truffle, Hardhat, or Remix.

Authority encrypts data and mints a zkSBT

Now, the *Authority* can encrypt the data of the *Identity holder*, mint a new zkSBT, and transfer it to that user.

The first step will be to encrypt the data of the *Identity holder*, using the *Identity holder*'s public key P, which must have been sent to the *Authority*. Once the *Authority* has the P public key, he will encrypt the data using any asynchronous encryption method. In our example, we have used the JavaScript Elliptic Curve Integrated Encryption Scheme (ECIES) library `ecies-geth`[11].

[11] https://www.npmjs.com/package/ecies-geth

```
function verifyProof(
    uint256 tokenId,
    uint[] memory proof,
    uint[] memory publicValues
) external view returns (bool) {
    address owner = address(uint160(publicValues[3]));
    require(
        ownerOf(tokenId) == owner,
        "The SBT doesn't belong to the address stored in the ZK proof"
    );
    bytes memory root = getRoot(tokenId);
    require(
        keccak256(abi.encodePacked(root)) ==
            keccak256(abi.encodePacked(publicValues[2])),
        "The root of the Merkle Tree's data doesn't match the root stored in the
            SBT"
    );

    uint[2] memory a = [proof[0], proof[1]];
    uint[2][2] memory b = [[proof[2], proof[3]], [proof[4], proof[5]]];
    uint[2] memory c = [proof[6], proof[7]];
    uint[7] memory p = [
        publicValues[0],
        publicValues[1],
        publicValues[2],
        publicValues[3],
        publicValues[4],
        publicValues[5],
        publicValues[6]
    ];

    require(_verifier.verifyProof(a, b, c, p),
      "Proof verification failed");
    return true;
}
```

Listing 39: verifyProof() function

```
function getRoot(
  uint256 tokenId
) public view override returns (bytes memory) {
  return sbtData[tokenId].root;
}

function getEncryptedData(
  uint256 tokenId
) external view override returns (bytes[] memory) {
  return sbtData[tokenId].encryptedData;
}
```

Listing 40: getRoot() and getEncryptedData() functions

In the Listing 41 we can see the Typescript code used to encrypt a piece of data using a public key.

```
encryptedCreditScore = await encryptWithPublicKey(
  identityHolder.publicKey,
  creditScore.toString()
);

encryptedIncome = await encryptWithPublicKey(
  identityHolder.publicKey,
  income.toString()
);

encryptedReportDate = await encryptWithPublicKey(
  identityHolder.publicKey,
  reportDate.toString()
);
```

Listing 42: Encrypt data using a public key

```
const ecies = require("ecies-geth");
const ethUtil = require("ethereumjs-util");

const toBuffer = (value: any) => {
  if (typeof value === "string") {
    return Buffer.from(value);
  } else {
    return Buffer.from(value.toString());
  }
};

const encryptWithPublicKey = async (publicKey: string, value: any) => {
  // Convert the public key to a buffer
  const publicKeyBuffer = ethUtil.toBuffer(publicKey);
  // Convert the value to a buffer
  const valueBuffer = toBuffer(value);

  // Encrypt the message
  const encryptedMessage = await ecies.encrypt(publicKeyBuffer, valueBuffer);

  return "0x" + encryptedMessage.toString("hex");
};
```

Listing 41: Function to encrypt data using a public key

Using this `encryptWithPublicKey` function, we can encrypt any kind of data. In our case, we will encrypt a sample `CreditScore`, `Income` and `ReportDate` values to be stored inside the zkSBT. This process can be viewed in Listing 42.

Another piece of information that we need to mint a new zkSBT is the Merkle tree root of the unencrypted data. In our case, we will use the ZK-friendly hashing algorithm *Poseidon*, using the `buildPoseidon` function of the `circomlibjs` library. The code to create this Merkle tree root can be viewed in Listing 43. Notice that the first piece of data stored in the Merkle tree is the address of the *Identity holder* (a_I) that will receive the zkSBT. This value is always added to prevent collisions in the Merkle tree root hashing function.

```
const buildPoseidon = require("circomlibjs").buildPoseidon;

const poseidon = await buildPoseidon();
const root = poseidon([
  BigInt(identityHolder.address),
  BigInt(creditScore),
  BigInt(income),
  BigInt(reportDate)
]);
const rootHex = "0x" + BigInt(poseidon.F.toString(root)).toString(16);
```

Listing 43: Get Merkle tree root

```
const hre = require("hardhat");
const ethers = hre.ethers;

const wallet = new ethers.Wallet('AUTHORITY_PRIVATE_KEY');
const authoritySigner = wallet.connect(provider);

const zkSBT = await ethers.getContractAt("ZKSBT", "CONTRACT_ADDRESS",
    authoritySigner);

await zkSBT
  .mint(identityHolder.address, rootHex, [
    encryptedCreditScore,
    encryptedIncome,
    encryptedReportDate
  ]);
```

Listing 44: Mint new zkSBT to *Identity holder*

Once we have the Merkle tree root of the unencrypted data $H(d)$, and all the encrypted data D, we can mint a new zkSBT sending it to the *Identity holder* address a_I. This process is listed in Listing 44. Once we have minted the new zkSBT, the *Identity holder* will own this token, and the encrypted data will be stored on-chain, together with the Merkle tree root of the unencrypted data.

Identity holder reads zkSBT and generates a ZK proof

When a *Verifier* user asks the *Identity holder* to check some information about his data, he needs to send a question containing this information:

- `index` (i) of the data that we want to check. This information is related to the position of the data inside the Merkle tree. In our example, 0 is the address of the *Identity holder*, 1 is the credit score, 2 is the income and 3 is the report date
- `operator` (o) that we want to apply
  - 0: equal (==)
  - 1: different (!=)
  - 2: greater (>)

```
const ecies = require("ecies-geth");
const ethUtil = require("ethereumjs-util");

const decryptWithPrivateKey = async (privateKey: string, valueHex: string) => {
  // Convert the private key to a buffer
  const privateKeyBuffer = ethUtil.toBuffer(privateKey);
  // Convert the value in hex to a buffer
  const valueBuffer = Buffer.from(valueHex.slice(2), "hex");

  // Decrypt the message
  const decryptedMessage = await ecies.decrypt(privateKeyBuffer, valueBuffer);
  return decryptedMessage.toString();
};
```

Listing 46: Function to decrypt data using the private key

  - 3: greater or equal (>=)
  - 4: less (<)
  - 5: less or equal (<=)

- `threshold` (t) to be compared with the data specified by the `index` (d_i)

For example, we can suppose that the *Verifier* asks the *Identity holder* that he wants to check that he has a credit score (`index = 1`) greater or equal (`operator = 3`) than 9 (`threshold = 9`).

To generate the proof, the first step that the *Identity holder* needs to do is to read the Merkle tree root and the encrypted data from the on-chain stored data. This process is listed in Listing 45.

```
const hre = require("hardhat");
const ethers = hre.ethers;

const zkSBT = await ethers.getContractAt("ZKSBT", "CONTRACT_ADDRESS");

const root = await zkSBT.getRoot(tokenId);
const encryptedData = await zkSBT.getEncryptedData(tokenId);
```

Listing 45: Read the data from zkSBT and decrypt it

Another step that needs to be done by the *Identity holder* is to decrypt the encrypted data with his private key `p`. To do this, we need a similar function to the listed in Listing 41, `encryptWithPublicKey`, but with the inverse direction. We have created the `decryptWithPrivateKey` to do this process, showing it on Listing 46.

The decryption process of the encrypted on-chain data using the `decryptWithPrivateKey` function is listed on Listing 47. This process is done off-chain. With this, the private data is not revealed.

To generate the proof, we have created a function called `genProof`, which is listed on Listing 48. This function will use the *snarkjs* library, which provides tools for working with zkSNARKs, and will be executed off-chain, as the `decryptWithPrivateKey` function, to avoid revealing confidential data. Two variables, `wasm_path` and `zkey_path`, are defined to hold the paths to the `.wasm` and `.zkey` files. These files are generated

```
const decryptedCreditScore = await decryptWithPrivateKey(
  identityHolder.privateKey,
  encryptedData[0]
);

const decryptedIncome = await decryptWithPrivateKey(
  identityHolder.privateKey,
  encryptedData[1]
);

const decryptedReportDate = await decryptWithPrivateKey(
  identityHolder.privateKey,
  encryptedData[2]
);
```

Listing 47: Decrypt data using the private key

```
const snarkjs = require("snarkjs");

const wasm_path = "circuits/verify4_js/verify4.wasm";
const zkey_path = "circuits/verify4.zkey";

const genProof = async (input) => {
  const { proof, publicSignals } = await snarkjs.groth16.fullProve(
    input,
    wasm_path,
    zkey_path
  );

  const solidityCallData = await snarkjs.groth16.exportSolidityCallData(
    proof,
    publicSignals
  );

  const argv = solidityCallData.replace(/["[\]\s]/g, "").split(",");

  const Proof = argv.slice(0, 8);
  const PubSignals = argv.slice(8);

  return { Proof, PubSignals };
};
```

Listing 48: genProof() function

during the compilation of the circuit and contain the compiled circuit and the proving key, respectively.

The `genProof` function takes an `input` parameter that contains all public and private inputs of the ZKP proof and returns an object containing two arrays, `Proof` and `PubSignals`. Inside the `genProof` function, the `snarkjs.groth16.fullProve` method is used to generate a proof and the associated public signals. This method takes three arguments: the `input` data, the path to the `.wasm` file, and the path to the `.zkey` file. The call of the `snarkjs.groth16.exportSolidityCallData` method and the

```
const proof = await genProof({
  index: 1, // credit score
  root: root,
  owner: identityHolder.address,
  threshold: 9,
  operator: 3, // 3 = greater or equal than
  timestamp: 1675195348511, // current timestamp
  value: +creditScore,
  data: [
    identityHolder.address,
    +creditScore,
    +income,
    +reportDate
  ]
});
```

Listing 49: call genProof() function

following lines are used to export the proof and public signals data in a format suitable for use in a Solidity smart contract, which are two unidimensional arrays.

Now the *Identity holder* can generate the proof by calling the `genProof` function, passing all necessary public and private arguments of the proof. This is listed on Listing 49. As we can see, the *Identity holder* will provide all the information asked by the *Verifier* (`index` of the data that he wants to query, with the `operator` and `threshold`), the `timestamp`, the `root` of the Merkle tree, stored in the zkSBT, the *Identity holder* address, and the private values, the value that we want to check d_i and the rest of the unencrypted data d.

As a result, we will have a variable `proof` that contains two unidimensional array elements, one containing the cryptographic proof of the ZKP (`Proof`) and another containing the public signals of the ZKP (`PubSignals`). These values can be sent to the *Verifier* user, who will check the validity of this proof.

Verifier verifies ZK proof

The last step is the verification process done by the *Verifier* user. This step is very simple because it only consists of getting the generated `proof`, and calling the `verifyProof` function of the zkSBT smart contract. This process is listed in Listing 50

```
const hre = require("hardhat");
const ethers = hre.ethers;

const zkSBT = await ethers.getContractAt("ZKSBT", "CONTRACT_ADDRESS");

const proofValid = await zkSBT.verifyProof(
  tokenId,
  proof.Proof,
  proof.PubSignals
);
```

Listing 50: Verify the proof

As we have described previously, this function will check:

- That the owner of the specified `tokenId` is the same as the address that is stored in the public values of the proof.
- That the Merkle tree root stored in the specified `tokenId` token is the same as the root stored in the proof.
- That the `verifier.sol` smart contract verifies correctly the proof. That is, the Merkle tree root stored in the proof (and in the smart contract) is the same as the Merkle tree root generated from the data without encryption (this means that the *Identity holder* hasn't modified the data), and the value stored at position `index` meets the condition `operator` over a `threshold`.

With the blockchain public information, the *Verifier* user can also be sure that the `tokenId` is minted by a specific *Authority* account and owned by a specific *Identity holder* account.

16.4 Security properties analysis

In this section, we will evaluate the security, privacy, and functionality aspects of the protocol. The properties under consideration include Integrity, Authenticity, Effectiveness, Fairness, Transferability of evidence, Timeliness, Timestamping, Non-repudiation, and Confidentiality, key attributes for a Private Identity-Related Attribute Verification Protocol. The security properties exclude the *efficiency* property for a distinct evaluation, which is conducted in section 16.5 that includes the presentation of results and a series of experiments aimed at assessing the protocol's performance.

1. **Integrity**.

   Proposition 1 *The identity-related attributes included in the zkSBT cannot be modified since the user attributes issued by trusted authorities cannot be counterfeited.*

   Claim 1 *The data included in the blockchain operations, such as the mint of a token, the identity of the Authority and the Identity holder, cannot be modified.*

   Proof: All blockchain transactions are stored in the blockchain. Immutability is an intrinsic feature of the blockchain. Then all the data included in the blocks cannot be modified.

   Claim 2 *Identity-related attributes are strongly linked to the zkSBT.*

   Proof: Identity-related attributes are used in the generation of the zkSBT. The root of a Merkle tree of these data in clear is stored in the zkSBT. This data cannot be modified since any change will influence the resulting value of the Merkle Tree root.

2. **Authenticity**.

   Proposition 2 *Both the origin of the token and its holder can be authenticated.*

   Claim 3 *The minter's identity of the token is publicly verifiable.*

   Proof: Due to the features of blockchain, anyone can check the blockchain address from which each zkSBT has been minted. This address must correspond to the Authority trusted by the verifiers to issue this kind of attribute. There is a public transaction on the blockchain from the Authority's address when executing the `Mint()` function in order to create any specific zkSBT.

   Claim 4 *The identity of the Holder of the token can be publicly verified.*

   Proof: Due to the features of blockchain, anyone can check the identifier of the Holder of the specific zkSBT. This is facilitated by a mapping within the smart contract to register the ownership information of each token. This mapping can be publicly verified.

3. **Effectiveness**.

   Proposition 3 *If all the actors follow the protocol, it is effective and the private identity-related attributes can be verified by the Verifier.*

   Claim 5 *The Effectiveness property is achieved through the meticulous integration of SBTs and ZKPs. The protocol demonstrates its efficacy in reliably and accurately verifying identity-related attributes while maintaining privacy and confidentiality.*

   Proof: The use of SBTs ensures secure and tamper-resistant identity binding, while ZKPs enable attribute validation without disclosing sensitive information. This seamless integration of components not only fulfills the protocol's objectives but also does so with exceptional effectiveness, establishing it as a robust solution for private identity verification.

   Proposition 4 *The fairness property arises in this protocol when one party (i.e. the Verifier) wants to trustfully check an attribute from an Identity holder while this user wants to proof that the attribute's value is correct without revealing it. Thus, on the one hand, the Verifier gets the proof and, on the other hand, the Identity holder does not need to reveal a private attribute.*

   Claim 6 *The protocol ensures equitable treatment of participants by incorporating mechanisms that prevent bias or favoritism in the attribute verification process.*

Proof: This fairness is maintained through transparent and unbiased procedures, assuring that all involved parties have an equal and fair experience within the identity verification framework. The utilization of SBTs and ZKPs contributes to the impartiality of the protocol, as these cryptographic techniques inherently prioritize privacy and prevent any undue advantage or discrimination during the verification process. Consequently, the implementation guarantees a fair and equitable environment for all entities involved in the identity-related attribute verification, fostering trust and reliability in the protocol.

Claim 7 *The protocol is fair because a Verifier, which seeks specific data from an Identity holder, requests whether an attribute has a value that is in accordance with a certain threshold or not and the protocol can assure him the correctness of that proof.*

Proof: The protocol assures the correctness of the proof because it is made using the data inside the zkSBT owned by the *Identity holder* and minted by the *Authority*. At the same time, the *Identity holder* does not have to reveal the attribute's value because it is encrypted inside the token but it can be properly verified using the prf proof generated by the identity holder and the function `verifyProof()` of the zkSBT smart contract deployed on the blockchain by the trusted *Authority*. Therefore, the protocol is fair because the *Verifier* can get what he wants (the verification of the attribute according to an established threshold) while the *Identity holder* preserves the confidentiality of his data. In this way, the protocol guarantees equitable treatment to the parties.

4. **Transferability of evidence**.

Proposition 5 *Evidence supporting identity-related attributes can be seamlessly transferred and utilized across different entities or systems, all the while ensuring its authenticity and integrity remain intact.*

Claim 8 *The use of SBTs and ZKPs enhances the transferability of evidence by allowing for secure and privacy-preserving sharing of relevant information. ZKPs, in particular, enable the verification of identity-related attributes without disclosing the actual information, ensuring that evidence can be presented without compromising the confidentiality of sensitive data.*

Proof: The cryptographic nature of SBTs further contributes to the secure transfer of evidence. These tokens, bound to the identity attributes, serve as verifiable credentials that can be presented across various interactions, establishing a consistent and reliable basis for attribute verification.

Therefore, the protocol's design, incorporating advanced cryptographic techniques, ensures that evidence supporting identity-related attributes can be effectively transferred between different entities or systems, meeting the criteria for the "Transferability of evidence" property in a secure and privacy-preserving manner.

5. **Temporal parameters**.

   Proposition 6 *The protocol achieves Reduced delay and Timestamping.*

   Claim 9 *The protocol ensures that the time required in the protocol operations is reduced by implementing efficient and prompt processes for attribute verification.*

   Proof: Through optimized algorithms and streamlined procedures, the protocol minimizes the time required to verify identity-related attributes. This is crucial in scenarios where swift verification is essential, such as access to time-sensitive services or applications. The protocol's design prioritizes minimizing processing delays while upholding the accuracy and security of attribute verification.

   Claim 10 *The inclusion of timestamping mechanisms within the protocol establishes a temporal reference for each verification transaction.*

   Proof: The timestamping of the blocks ensures a clear chronological order of events, aiding in auditing, accountability, and dispute resolution. The timestamping feature, combined with the cryptographic assurances of ZKPs and SBTs, enhances the overall reliability of the protocol by providing a verifiable record of when attribute verifications occurred.

6. **Non-repudiation**.

   Proposition 7 *The protocol provides both non-repudiation of origin and non-repudiation of reception in all the involved deliveries.*

   The accomplishment of the "Non-repudiation" property in the implementation of the Private Identity-Related Attribute Verification Protocol using SBTs and ZKPs is grounded in the cryptographic and secure design of the protocol, ensuring that entities involved in the verification process cannot deny their actions or involvement.

   Claim 11 *The protocol provides non-repudiation of data origin.*

   Proof: Anyone can check who minted a zkSBT. Thus, the origin of the data inside each token has been produced by the entity that generates the token. If this entity is not a trusted authority, then the validity of the data can be put under question. However, the entity that minted the token cannot reject the origin of the data since there is a signed transaction from its blockchain address to create the new zkSBT.

   Claim 12 *The protocol provides non-repudiation of delivery.*

Proof: In this scenario, we understand by non-repudiation of delivery the fact that the *Identity holder* cannot deny having generated the prf proof of the ZKP procedure in order to create a piece of evidence to demonstrate if the requested attribute associated with his identity meets the threshold requirement or not. This prf proof has been delivered to the *Verifier*. After receiving prf, the Verifier calls the `verifyProof()` function of the zkSBT. The smart contract checks that the owner of the token is the intended holder and, if prf meets the requirements of the criteria outlined in the circuit, then the authenticity of the *Identity holder* is proven, since only the genuine owner of the token can generate a proper prf. However, if the verification fails because the value of prf does not meet the established requirement, then we need to add a secure communication channel between *Identity holder* and *Verifier* that guarantees the non-repudiation of delivery of the message containing the prf item. Nevertheless, the addition of a secure channel (e.g. a TLS secure channel[12]) does not change the specification of the protocol proposed in this chapter.

Claim 13 *The protocol provides non-repudiation evidence of reception of the verification results.*

Proof: The *Verifier*, after receiving the proof from the *Identity holder*, is not able to get any kind of acknowledgment of the value of the attribute if it does not execute the `verifyProof()` of the zkSBT smart contract. This function will provide the result of the verification that will be available. The verification process also includes the attestation of the token owner and the token issuer. Since there is a signed transaction from the *Verifier* address to run the `verifyProof()` function, then this user cannot deny having received the verification results because they are available on the blockchain due to the execution of the transaction the *Verifier* submitted.

7. **Confidentiality**.

Proposition 8 *The confidentiality of identity-related attributes is maintained even when some of its properties are proven to the Verifier.*

Claim 14 *Data contained in the zkSBTis never revealed to the Verifier of the identity-related attributes.*

Proof: The encrypted identity-based attributes are stored in the token. The encryption is performed using the public key of the *Identity holder*, ensuring that only the *Identity holder* can decrypt them using its private key. Although the verifier can check if the values of these attributes fulfill specific requirements, the values themselves remain confidential.

[12]https://www.ncsc.gov.uk/guidance/using-tls-to-protect-data

16.5 Performance analysis

After incorporating this protocol's implementation, we have evaluated its performance using the `hardhat-gas-reporter` plugin[13], an integral component of the Hardhat development environment. This plugin enables the calculation of Gas usage per unit test by leveraging metrics for method calls and deployments.

We have created some tests to verify the accuracy of this protocol correction. These tests, coupled with the aforementioned `hardhat-gas-reporter` plugin, enable us to assess the system's efficiency in terms of gas cost.

Gas execution costs play a crucial role in Ethereum Virtual Machine (EVM) compatible blockchains, the platforms on which we execute the developed smart contracts. These costs can be a significant consideration in the development of this service.

To assess the gas cost in US Dollars (USDs), we have used the Polygon PoS network, an Ethereum layer 2 scaling solution designed for faster and more economical transactions. Polygon PoS generally incurs lower gas costs compared to Ethereum due to its distinct fee structure. Furthermore, being a layer 2 scaling solution, Polygon PoS consolidates transactions and settles them on the Ethereum mainnet in a single transaction, effectively lowering the overall gas expenditure. In summary, opting for Polygon PoS can yield substantial cost savings for users engaging in frequent or high-value transactions, presenting an appealing alternative to Ethereum for those seeking gas cost reductions.

Based on the test results, we have analyzed the gas cost associated with the deployment of `verifier.sol` and `ZKSBT.sol` smart contracts, and the `mint` function, the only one that has a gas cost, because the rest of the functions are "view" functions, that is, read-only. View functions in Ethereum Virtual Machine (EVM) blockchains do not cost gas when they are called externally. This is because they are simply reading from the blockchain state and not modifying it. This is the case of the `verifyProof` function, which doesn't have any cost.

Figure 16.4 depicts these costs as reported by the `hardhat-gas-reporter` plugin. It's essential to note that these costs are measured in gas units. To check the precise transaction price, we need data on the current gas price in MATIC (the native currency

[13]https://www.npmjs.com/package/hardhat–gas–reporter

Methods	108 gwei/gas	0.86 usd/matic
Contract	**Method**	**Avg**	**usd (avg)**
ZKSBT	mint	574668	0.05
Deployments			
Verifier		960115	0.09
ZKSBT		1889065	0.18

Figure 16.4: Gas cost of the ZKSBT and Verifier smart contracts

Table 16.2: Average costs of the ZKSBT and Verifier smart contracts

	USD
ZKSBT deployment	0.36712
Verifier deployment	0.18814
ZKSBT.mint()	0.11168

of the Polygon PoS) and the MATIC to USDs exchange rate. For this evaluation, we referenced the MATIC price as of December 2023, which stands at 0.86 USD/MATIC. This dollar reference is for illustrative purposes only, calculated at the time of the study. For comparing different results, it is more accurate to compare gas costs directly (Figure 16.5).

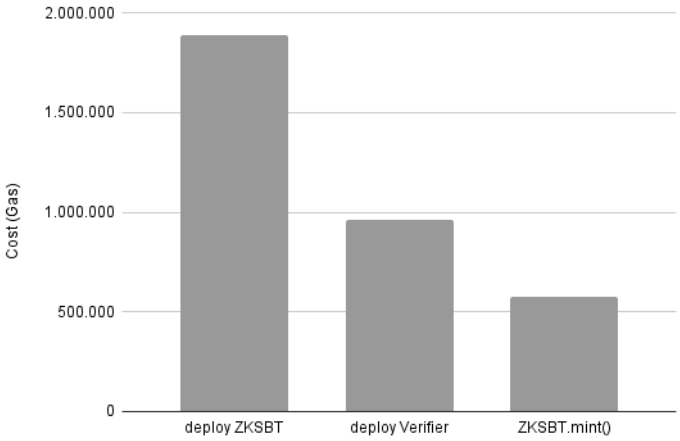


Figure 16.5: Gas cost comparison of the ZKSBT and Verifier deployments and mint function

Analyzing the gas cost and average gas cost (see Table 16.2) associated with both smart contract deployment and function execution, it becomes evident that the most expensive function, in terms of gas cost, is the deployment of the `ZKSBT.sol` smart contract itself. This incurs a cost of 0.18 USD during our test executions and maintains an average cost of 0.37 USD over the past year. The deployment of the `verifier.sol` smart contract is cheaper, with an average cost of 0.19 USD, and the `mint()` function is relatively economical, with an average cost of 0.11 USD. It is necessary to note that in the long term, the cost of the `mint()` function is the most important, since once deployed all smart contracts, we only need to call this minting function.

Additionally, it is important to note that the cost of the `mint()` function depends significantly on the attribute set for the zkSBT being minted. In the specific case of the report, the attributes are `creditScore`, `income`, and `reportDate`, with values '45', '3100', and '2023-01-31T20:23:01.804Z' respectively. These details can be verified in the test section of the protocol's GitHub repository. The resultant gas cost of the `mint()` function will vary depending on the number and length of attributes added.

Taking into account these findings, in conjunction with a noticeable spike in gas costs detected in mid-November 2023, the expenses linked to the specified and implemented protocol are reasonably budget-friendly, aligning well with the desired attributes for managing a Private Identity-Related attribute verification system.

16.6 Conclusions

The integration of public-key cryptography, SBTs, and ZKPs holds the promise to transform our approach to SSI and data privacy. Empowering individuals with enhanced control over their personal data and facilitating secure and private data sharing in a decentralized fashion, these technologies will foster the development of a trust-driven digital society.

In the identity-related attribute verification protocol that we propose for privacy preservation, we've illustrated the utilization of zkSBTs to represent and verify attributes related to personal identity. Additionally, ZKPs facilitate secure and private data sharing, eliminating the reliance on centralized authorities. Through the integration of these technologies, we can establish a system that empowers individuals to retain control over their personal data, selectively sharing it with trusted parties while ensuring both privacy and security.

zkSBTs have a variety of potential use cases in the context of SSI and digital asset management. Here are a few examples:

1. **Secure Identity Verification**: zkSBTs can be used to enable secure and private identity verification without the need for a central authority or third-party verification service. For example, an authority could create a zkSBT that represents an identity, and then, the holder of that zkSBT, use a ZKP to prove their identity to a third party without revealing any additional personal information.

2. **Selective Data Sharing**: zkSBTs can be used to enable selective data sharing in an SSI system. For example, an authority could create a zkSBT that represents the age of a user, and then the user could use a ZKP to prove their age to a third party without revealing their exact date of birth or any other personally identifying information. Other data-sharing examples are: credit scoring, voting rights, reputation systems, and medical data.

3. **Secure Access Control**: zkSBTs can be used to enable secure access control to digital resources, such as websites or online services. For example, we can create a zkSBT that represents an authorization to access a particular online service, and use the token to securely authenticate their access without the need for usernames, passwords, or other traditional authentication methods.

The zkSBT protocol has been adopted by **Masa**[14], a company leveraging this technology to power privacy-preserving data exchange. This adoption further validates the practical applicability and effectiveness of the protocol in real-world financial services. For more details, refer to Masa's announcement[15].

[14] https://masa.ai

[15] https://medium.com/masa-finance/masa-invents-the-zksbt-to-power-privacy-preserving-data-exchange-9159cf72ff10

Overall, zkSBTs have the potential to greatly enhance the security and privacy of SSI systems and digital asset management, by enabling selective data sharing, secure identity verification, and decentralized asset management, all while maintaining individual control over personal data.

The integration of public-key cryptography, SBTs, and ZKPs holds the promise to transform our approach to SSI and data privacy. Empowering individuals with enhanced control over their personal data and facilitating secure and private data sharing in a decentralized fashion, these technologies will foster the development of a trust-driven digital society.

In the identity protocol that we propose for privacy preservation, we've illustrated the utilization of zkSBTs to represent attributes related to personal identity. Additionally, ZKPs facilitate secure and private data sharing, eliminating the reliance on centralized authorities. Through the integration of these technologies, we can establish a system that empowers individuals to retain control over their personal data, selectively sharing it with trusted parties while ensuring both privacy and security.

Moreover, it utilizes decentralized and distributed ledger technologies, such as blockchain, to create trust and transparency in identity management. Transactions and interactions documented on a blockchain are subject to audit and verification, fostering trust between parties and diminishing the risk of fraud.

In this chapter we also have seen that the costs associated with the specified and implemented protocol are reasonably affordable, aligning seamlessly with the desired characteristics for managing a system for Private Identity-Related attribute verification. We also have demonstrated that thanks to the inherent characteristics of the technologies used in this protocol, we acquire key security properties.

Deploying the zkSBT protocol at scale involves several challenges that need to be addressed to ensure its widespread adoption and effectiveness:

- Blockchain networks like Ethereum, while secure, can face scalability issues when handling a large number of transactions simultaneously. This can lead to increased transaction times and higher costs. To solve this, Layer 2 scaling solutions, such as Polygon, can be used to alleviate network congestion and reduce transaction costs. Additionally, ongoing developments in blockchain technology, such as Ethereum 2.0, aim to enhance scalability.

- Generating and verifying Zero-Knowledge Proofs can be computationally intensive, potentially slowing down the verification process. However, optimization techniques in ZKP algorithms and the use of efficient ZKP frameworks like zkSNARKs can reduce computational overhead. Research into more efficient ZKP protocols, such as Zero-Knowledge Scalable Transparent Argument of Knowledges (zkSTARKs), is ongoing and shows promise for further improvements.

- Ensuring that users can easily interact with the protocol and understand the privacy benefits without being overwhelmed by the technical complexities. To solve this, is important to develop intuitive user interfaces, such as the prototype UI implemented in our project, and provide clear, user-friendly documentation and support that can enhance user adoption.

In considering future work for the implementation of a Private Identity-Related Attribute Verification Protocol using SBTs and ZKPs, we think that several options merit exploration to enhance the protocol's functionality and address emerging challenges:

- **Revocation mechanism for identity**. To fortify the protocol's adaptability, a crucial aspect of future development involves exploring mechanisms for identity revocation. Implementing a system that allows authorized entities to revoke an individual's identity under specific circumstances would further enhance the security and flexibility of the protocol.

- **Implementation of data expiration**. To enhance the temporal control and privacy of the data shared through SBTs, a valuable option for future work involves the incorporation of an expiration date mechanism. This feature would enable authorities to set time limits on the validity of the data associated with SBTs sent to *Identity holders*. Introducing this capability would not only align with evolving privacy regulations but also provide individuals with greater autonomy over the lifecycle of their shared information, contributing to the overall robustness of the protocol.

- **Enhanced token transfer control**. The protocol could benefit from an extended capability to reject the transfer of SBTs. Integrating the Rejectable NFT implementation, as outlined in Chapter 7, would improve the protocol by providing identity holders with the ability to exercise control over the transfers of the zkSBTs sent by authorities, aligning with evolving user preferences and security needs.

- **Incorporation of off-chain data**. Enabling the addition of off-chain data represents an essential avenue for future work. Investigating methods to seamlessly integrate off-chain data, whether through decentralized solutions like IPFS or centralized databases, would expand the protocol's utility. This enhancement would empower users to include a bigger amount of information while maintaining the privacy and security afforded by the existing protocol.

- **Utilizing EIP-4337 standard for mass adoption**. To promote mass adoption of the protocol, exploring compatibility with emerging standards is pivotal. Integrating the EIP-4337 standard, which encompasses account abstraction and introduces paymasters capable of sponsoring transactions for users, could significantly enhance accessibility. This approach allows users to engage with the protocol without directly managing transaction costs, potentially fostering widespread adoption.

In summary, these proposed avenues for future work aim to fortify the security, flexibility, and usability of the Private Identity-Related Attribute Verification Protocol, positioning it as a robust and adaptable solution in the rapidly evolving landscape of digital identity and privacy technologies.

The potential advantages of a privacy-focused identity protocol employing public-key cryptography, SBTs, and ZKPs are vast. Ranging from secure and private identity verification to decentralized management of digital assets and controlled data sharing, these technologies have the potential to foster a more fair and reliable digital

environment. Consequently, we emphasize the importance of ongoing research and development in this field for the advancement of privacy and digital security in the future.

CHAPTER 17

CONCLUSIONS

This chapter synthesizes the findings from the thesis, providing a critical evaluation of the research and its implications for the future of e-commerce and blockchain technology. By addressing the research questions and objectives, this chapter contributes to a deeper understanding of the challenges and opportunities presented by the integration of blockchain in e-commerce.

17.1 Thesis summary

The thesis has explored the potential of blockchain technology in revolutionizing e-commerce by enhancing security, transparency, privacy, and efficiency in online transactions. Initially, it reviews the limitations and challenges inherent in traditional e-commerce systems, such as security vulnerabilities, lack of transparency, and over-reliance on centralized third-party intermediaries.

The research delves into the unique attributes of blockchain technology, such as decentralization, immutability, and transparency, and assesses their applicability in addressing the aforementioned challenges in e-commerce. It outlines the design, development, and evaluation of novel blockchain-based e-commerce protocols aimed at overcoming these limitations. These protocols leverage smart contracts, cryptographic proofs, and blockchain's inherent properties to facilitate secure, transparent, and efficient online transactions.

Enumerated within this thesis are comprehensive analyses and developments of various blockchain protocols for e-commerce, including:

1. **Certified Notifications Protocols**: Both two-party and multiparty formats, which secure digital notifications without the need for Trusted Third Partys, enhancing efficiency and privacy.

2. **Rejectable NFTs Protocol**: Introduces a user-centric approach for asset management by allowing the rejection or acceptance of tokens.

3. **Contract Signing Protocols**: Utilize blockchain to execute agreements securely and efficiently, ensuring non-repudiation in B2B and B2C transactions.
4. **Micropurchases Using Payment Channels Protocol**: Offers a scalable solution for microtransactions, reducing costs and processing times.
5. **Decentralized and Confidential Digital Identity Protocol Using ZKP**: Ensures privacy-preserving identity verification, allowing users to prove identity attributes without revealing personal data.

The study presents the implementation of smart contracts for different e-commerce scenarios, offering insights into their real-world applicability, technical challenges, and user acceptance.

Additionally, the thesis discusses the practical implications of integrating blockchain technology into e-commerce, highlighting the existing gap between technological capabilities and practical implementation. It acknowledges the need for user-friendly interfaces, educational initiatives, and regulatory frameworks to bridge this gap and facilitate wider adoption.

In summary, this thesis contributes to the academic and practical understanding of how blockchain technology can be harnessed to advance e-commerce protocols, making online transactions more secure, transparent, and efficient. It sets the foundation for future research in this rapidly evolving field, pointing out potential areas for improvement and further investigation. The exploration of these protocols underscores the transformative potential of blockchain in e-commerce, while also highlighting significant challenges that remain, including user accessibility, interoperability between blockchain platforms, and the complexity of large-scale implementation.

17.2 Discussion

The integration of blockchain technology within e-commerce platforms has demonstrated significant improvements in transactional transparency, security, and efficiency. This study's findings reveal how blockchain can mitigate common e-commerce challenges such as fraud, data breaches, and lack of trust among parties. By providing a decentralized ledger for transactions, blockchain technology promotes an environment where all parties can verify transactional data directly, reducing the potential for discrepancies and enhancing the overall trustworthiness of online markets.

However, while blockchain presents a robust solution, the research has also highlighted the challenges of integrating such a decentralized system, particularly regarding scalability, economic cost of transactions, user adoption, and regulatory compliance. The transition from centralized systems, which many businesses rely on due to their established infrastructure and ease of use, to decentralized blockchains, introduces technical, organizational, and legal hurdles. The study discusses strategies for overcoming some of these barriers, such as developing hybrid models that combine the best features of both centralized and decentralized systems, or decreasing economic costs with different technical approaches.

Based on the comprehensive review of blockchain protocols and their implications for e-commerce within the thesis, the integration of blockchain technology into e-commerce protocols has shown to have profound impacts on enhancing transactional

security, efficiency, and user privacy. Through the detailed examination of various protocols, including two-party and multiparty certified notifications, rejectable NFTs, contract signing protocols, micropurchases using payment channels, and identity-related attribute verification protocol using SBTs and ZKPs, the study offers valuable insights into the practical applications and challenges of blockchain in e-commerce.

The development and analysis of certified notifications protocols, both in two-party and multiparty contexts, reveal the capabilities of blockchain to streamline and secure the process of digital notifications. These protocols eliminate the need for Trusted Third Partys (TTPs), enhancing the efficiency and integrity of electronic deliveries. The confidential variants of these protocols further ensure the privacy of communications, a critical aspect in business transactions and personal exchanges alike.

Rejectable NFTs introduce a novel approach to ownership and transferability, offering users the ability to reject or accept tokens, thereby providing a mechanism for more user-centric asset management. This protocol signifies a departure from traditional Non-Fungible Token standards by adding an extra layer of user consent, potentially revolutionizing how digital assets are exchanged and controlled in online marketplaces.

The exploration of contract signing protocols on the blockchain demonstrates a secure and efficient framework for executing agreements without the necessity for intermediaries. By leveraging the immutable and transparent nature of blockchain, these protocols offer a robust solution for facilitating binding agreements with non-repudiation, a key requirement in both B2B and B2C e-commerce settings.

Micropurchases using payment channels protocol showcases a scalable solution for handling small transactions, a common challenge in e-commerce. This protocol reduces transaction fees and processing times, making it viable for a wide range of microtransaction-based business models, from digital content purchases to IoT applications.

Lastly, the identity-related attribute verification protocol using SBTs and ZKPs addresses one of the most pressing issues in online interactions: privacy-preserving identity verification. This protocol enables users to prove their identity or the possession of specific attributes without disclosing the actual data, a significant advancement in protecting user privacy online.

These protocols collectively underscore the transformative potential of blockchain in enhancing the foundational aspects of e-commerce, from transaction processing to user privacy and digital identity verification. However, the findings also highlight the existing challenges, such as scalability, user adoption, and regulatory compliance, that need to be addressed to fully realize the benefits of blockchain technology in e-commerce.

The exploration of blockchain protocols within this thesis underscores their transformative capabilities for e-commerce, which herald notable improvements in transactional efficiency and the safeguarding of user privacy. The in-depth analysis reveals that these protocols hold considerable promise for revolutionizing e-commerce practices by streamlining operations and enhancing the security framework. Nonetheless, the research also delineates significant challenges that impede their widespread implementation. These include the need for enhanced user accessibility, ensuring seamless interoperability across diverse blockchain platforms, and addressing the complexities associated with large-scale deployment. Addressing these challenges is crucial for harnessing the full potential of blockchain technologies in e-commerce, pointing towards

the necessity for ongoing innovation and strategic collaborations to overcome these barriers and unlock the comprehensive benefits these protocols offer.

The study has extensively explored security and privacy considerations, which are essential in e-commerce settings. The implementation of smart contracts and Zero-Knowledge Proofs has been demonstrated to enhance data integrity and privacy, allowing for secure and confidential transactions without revealing unnecessary information. However, the research also points out the inherent risks associated with smart contract vulnerabilities and the challenges in achieving a balance between transparency and privacy. This leads to a discussion on the importance of ongoing security audits, the development of best practices for smart contract design, like the use of Factory Clone programming pattern used in Chapter 11, and the potential for new cryptographic methods to further enhance security without compromising efficiency.

The practical implications of applying blockchain technology in real-world e-commerce scenarios have been a significant focus of this thesis. The development and testing of smart contracts for various e-commerce protocols have provided valuable insights into their real-world applicability and user acceptance. While the results are promising, the discussion acknowledges the gap between technological capability and practical implementation, highlighting the need for user-friendly interfaces, and educational initiatives to facilitate wider adoption. Additionally, this segment explores the economic and social implications of widespread blockchain adoption in e-commerce, such as the potential for reduced transaction fees, increased market access for small and medium-sized enterprises, and greater consumer protection and empowerment.

17.3 Contributions

This thesis has made significant contributions to the field of blockchain technology, particularly in the realm of e-commerce protocols. The key contributions are as follows:

1. **Development of Blockchain-based E-commerce Protocols**: The thesis presents a comprehensive study on the integration of blockchain technology into e-commerce systems. By designing and implementing innovative blockchain-based solutions, this research addresses various e-commerce challenges such as fraud, transparency, and efficiency.

2. **Analysis of Blockchain Protocols for E-commerce**: An extensive evaluation of different blockchain protocols has been conducted, focusing on their suitability for e-commerce applications. This includes a detailed examination of payment channel networks, NFTs, and smart contracts, contributing to a better understanding of how these technologies can enhance e-commerce platforms.

3. **Security and Privacy Considerations**: This research has contributed to the security and privacy aspects of blockchain in e-commerce by implementing and assessing the effectiveness of smart contracts and zero-knowledge proofs. These mechanisms are crucial for maintaining data integrity and privacy in online transactions, representing a significant step forward in developing secure e-commerce platforms.

4. **Real-world Application and User Acceptance**: The thesis goes beyond theoretical analysis by exploring the practical implications of applying blockchain technology in real-world e-commerce scenarios. This includes the development, testing, and evaluation of smart contracts for various e-commerce protocols, providing valuable insights into their applicability and user acceptance. Notably, the collaboration with **Masa**[1] for the implementation and deployment of zkSBT smart contracts within the "Decentralized and Confidential Digital Identity Protocol Using ZKP" underscores the practical application and industry relevance of the research findings.

5. **Addressing Scalability and Interoperability Challenges**: The research identifies and proposes solutions to key challenges in integrating blockchain with existing e-commerce systems, such as scalability and interoperability. This contributes to the ongoing discussion on how to effectively adopt blockchain technology in mainstream e-commerce operations.

6. **Future Directions for Blockchain in E-commerce**: Finally, the thesis outlines potential future research areas, emphasizing the importance of user-friendly interfaces, educational initiatives, and regulatory frameworks. This sets the stage for further advancements in the field and encourages a multidisciplinary approach to overcoming the current limitations of blockchain in e-commerce.

These contributions not only advance the academic understanding of blockchain in e-commerce but also offer practical insights and solutions for businesses looking to leverage blockchain technology to improve their online platforms. The partnership with **Masa**, particularly, highlights the thesis's impact on bridging theoretical research with practical, real-world applications and deployments, emphasizing the potential of blockchain technology to solve critical issues in digital identity verification and privacy.

17.4 Limitations

While this thesis has provided significant insights into the integration of blockchain technology within e-commerce, there are several limitations to the study that should be acknowledged:

1. **Scalability and Performance**: The blockchain protocols developed and tested within this research demonstrate potential for improving e-commerce transactions' security and transparency. However, scalability remains a major concern. The high transaction costs and slow processing times associated with popular blockchain platforms like Ethereum could hinder widespread adoption, particularly during peak times.

2. **User Adoption and Experience**: The thesis assumes a level of familiarity and comfort with blockchain technology that may not exist among all e-commerce users and businesses. The complexity and technical nature of blockchain could deter widespread adoption without significant educational efforts and user-friendly interface designs.

[1] https://masa.ai

3. **Regulatory and Legal Challenges**: While the thesis explores the technical feasibility of blockchain protocols in e-commerce, it does not delve deeply into the regulatory and legal challenges. The lack of standardized regulations across different jurisdictions could pose significant challenges to implementing these solutions on a global scale.

4. **Integration with Existing Systems**: The research primarily focuses on blockchain's potential in isolation. In practice, integrating blockchain solutions with existing e-commerce platforms and payment systems will present technical and operational challenges not fully addressed in this study.

5. **Privacy Concerns**: Despite blockchain's potential for enhancing transaction security, privacy concerns arise, particularly in public blockchain networks where transaction details are transparent. The balance between transparency and privacy remains a complex issue that requires further exploration.

6. **Middleware Dependence**: The proposed protocols, particularly those involving confidential notifications and contract signings, rely on middleware for certain functionalities. This dependence could introduce a single point of failure and might contradict the decentralized ethos of blockchain.

7. **Real-world Identity Verification**: The thesis does not fully address the challenge of linking digital blockchain identities to real-world identities, which is crucial for many e-commerce transactions, particularly in legal and regulatory contexts.

8. **Technological Assumptions**: The study's findings are based on the current state of blockchain technology and might not hold if significant technological advancements or changes occur in the blockchain landscape.

This thesis has addressed and resolved many of the acknowledged limitations of blockchain technology, thereby enhancing its practical applicability and effectiveness. By identifying these gaps and providing solutions, the research laid out in this thesis sets a strong foundation for future studies to build upon, aiming to further refine and expand upon these advancements.

17.5 Future work

The chapter concludes with recommendations for future research, emphasizing areas such as cross-chain interoperability, privacy-preserving mechanisms, and the development of more scalable blockchain solutions. The potential for integrating emerging technologies like AI and IoT with blockchain for e-commerce is also discussed, suggesting a multidisciplinary approach for future studies. Building upon the current research, the following future directions are recommended:

1. **Rust Implementation of Proposed Algorithms**: Future studies could explore implementing the proposed blockchain protocols using Rust, a programming language known for its safety and performance features. This could particularly be beneficial for protocols where security and speed are paramount. Additionally,

examining the feasibility of deploying these algorithms on the Solana blockchain, known for its high throughput and lower transaction costs, could address some of the scalability issues identified in this thesis.

2. **Use of EIP-4337 Account Abstraction**: The implementation of EIP-4337[183] could be explored in future research to simplify user interactions with blockchain-based e-commerce platforms. Account abstraction could allow for more user-friendly transaction processes, potentially increasing blockchain technology's adoption among non-technical e-commerce users. Notable additions of this standard include gasless transactions, account recovery facilitated by multi-signature wallets, as well as the integration of social recovery features.

3. **Integration of Blockchain-Based Identity Systems**: Current protocols identify users solely by their blockchain addresses. Future research should explore the integration of blockchain-based identity systems, which would enable interactions based on verified user identities rather than anonymous addresses. This integration could enhance security and trust in blockchain transactions, facilitating more personalized and legally compliant interactions across various platforms. Such systems could include identity verification (KYC processes) used on Centralized Exchanges, or leveraging self-sovereign identity systems that use Decentralized Identifiers (DIDs) on the blockchain, providing a reliable method to associate users with their accounts.

4. **Exploration of IPFS for Encrypted Data**: Future work will delve into a more detailed examination of using the InterPlanetary File System (IPFS) for encrypted data storage in blockchain networks. This research will address potential vulnerabilities associated with the public sharing of sensitive information, aiming to reinforce data security while maintaining system integrity.

5. **Adoption of the Foundry Framework**: Future work should consider utilizing the Foundry framework for testing, deploying, and compiling smart contracts. This framework can provide a more efficient and streamlined development environment, facilitating the creation of robust and secure blockchain protocols for e-commerce.

6. **Cross-Chain Interoperability**: Further research should focus on enhancing cross-chain interoperability to enable seamless transactions across different blockchain networks. This would address the current limitations related to platform dependency and promotes a more integrated and efficient blockchain ecosystem for e-commerce.

7. **Privacy-Preserving Mechanisms**: Continuing the development of privacy-preserving mechanisms is crucial. Future research should delve deeper into the integration of advanced cryptographic techniques, such as Zero-Knowledge Proofs and homomorphic encryption, to balance transparency and privacy in blockchain-based e-commerce.

8. **Scalability of Blockchain Solutions**: Addressing the scalability challenges identified in this thesis is critical for the future success of blockchain in e-commerce.

Research should focus on developing more scalable blockchain solutions that can handle large volumes of transactions without compromising speed or increasing costs.

9. **Integration with Artificial Intelligence (AI) and Internet of Things (IoT)**: The integration of blockchain with other emerging technologies like Artificial Intelligence (AI) and the Internet of Things (IoT) presents a promising route for future research. These integrations could lead to more intelligent, automated, and personalized e-commerce experiences, further enhancing efficiency and user satisfaction.

By focusing on these areas, future research can build upon the foundations laid by this thesis, addressing its limitations and contributing to the continued evolution and application of blockchain technology in the e-commerce sector.

BIBLIOGRAPHY

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf 1.1

[2] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking digital cryptocurrencies*. O'Reilly Media Inc., 2017. 1.1

[3] A. M. Antonopoulos and G. Wood, *Mastering Ethereum: Building Smart Contracts and Dapps*. O'Reilly Media Inc., 2019. 1.1, 4.4, 4.7.3, 6

[4] M. Swan, *Blockchain: Blueprint for a New Economy*. O'Reilly Media, 2015. 1.1

[5] N. D. Bhaskar and K. C. Lee, "Bitcoin mining technology," *Handbook of Digital Currency: Bitcoin, Innovation, Financial Instruments, and Big Data. Research Collection Lee Kong Chian School Of Business*, pp. 45–65, 2015. 4.2

[6] J. Zou, B. Ye, L. Qu, Y. Wang, M. A. Orgun, and L. Li, "A proof-of-trust consensus protocol for enhancing accountability in crowdsourcing services," *IEEE Transactions on Services Computing*, vol. 12, no. 3, pp. 429–445, 2019. 4.2

[7] M. Kuhn and J. Franke, "Data continuity and traceability in complex manufacturing systems: a graph-based modeling approach," *International Journal of Computer Integrated Manufacturing*, vol. 34, no. 5, pp. 549–566, 2021. 4.2

[8] J. Sunny, N. Undralla, and V. Madhusudanan Pillai, "Supply chain transparency through blockchain-based traceability: An overview with demonstration," *Computers & Industrial Engineering*, vol. 150, p. 106895, 2020. 4.2

[9] F. Calvão and M. Archer, "Digital extraction: Blockchain traceability in mineral supply chains," *Political Geography*, vol. 87, p. 102381, 2021. 4.2

[10] X. Yang, M. Li, H. Yu, M. Wang, D. Xu, and C. Sun, "A trusted blockchain-based traceability system for fruit and vegetable agricultural products," *IEEE Access*, vol. 9, pp. 36 282–36 293, 2021. 4.2

[11] N. Bozic, G. Pujolle, and S. Secci, "A tutorial on blockchain and applications to secure network control-planes," *2016 3rd Smart Cloud Networks & Systems (SCNS). Institute of Electrical and Electronics Engineers Inc.*, pp. 1–8, 2016. 4.3

[12] M. Vukolić, "Rethinking permissioned blockchains," *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts, co-located with ASIA CCS*, pp. 3–7, 2017. 4.3

[13] M. Salimitari, M. Chatterjee, and Y. Fallah, "A survey on consensus methods in blockchain for resource-constrained iot networks," *Internet of Things (Netherlands). Elsevier B.V.*, 2020. 4.3

[14] M. Ahmed, S. Reno, N. Akter, and F. Haque, "Securing medical forensic system using hyperledger based private blockchain," *2020 23rd International Conference on Computer and Information Technology (ICCIT)*, pp. 1–6, 2020. 4.3

[15] C. Ma, X. Kong, Q. Lan, and Z. Zhou, "The privacy protection mechanism of hyperledger fabric and its application in supply chain finance," *Cybersecurity*, vol. 2, no. 1, pp. 1–9, 2019. 4.3

[16] Z. Shahbazi and Y.-C. Byun, "Integration of blockchain, iot and machine learning for multistage quality control and enhancing security in smart manufacturing," *Sensors*, vol. 21, no. 4, pp. 1–21, 2021. 4.3

[17] M. Walker, "Distributed ledger technology: Hybrid approach, front-to-back designing and changing trade processing infrastructure," 2018. 4.3

[18] A. Singh, K. Click, R. M. Parizi, Q. Zhang, A. Dehghantanha, and K.-K. R. Choo, "Sidechain technologies in blockchain networks: An examination and state-of-the-art review," *Journal of Network and Computer Applications. Academic Press*, vol. 149, p. 102471, 2020. 4.3

[19] G. Wood, "Ethereum: yellow paper," 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf 4.4, 14.4

[20] C. Dannen, *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. Apress, 2017. 4.4

[21] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, "Smart contract development: Challenges and opportunities," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2084–2106, 2021. 4.4

[22] M. Bez, G. Fornari, and T. Vardanega, "The scalability challenge of ethereum: An initial quantitative analysis," *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), San Francisco, CA, USA*, pp. 167–176, 2019. 4.5

[23] M. Schäffer, M. Di Angelo, and G. Salzer, "Performance and scalability of private ethereum blockchains," *Lecture Notes in Business Information Processing. Springer Verlag*, vol. 361, pp. 103–118, 2019. 4.5

[24] V. Buterin, "Ethereum scalability research and development subsidy programs," 2018. [Online]. Available: https://blog.ethereum.org/2018/01/02/ethereum-scalability-research-development-subsidy-programs 4.5

[25] ——, "An incomplete guide to rollups," 2021. [Online]. Available: https://vitalik.ca/general/2021/01/05/rollup.html 4.5

[26] M. William, "Erc-20 tokens, explained," *Cointelegraph*, 2018. [Online]. Available: https://cointelegraph.com/explained/erc-20-tokens-explained 4.6

[27] A. Arora, Kanisk, and S. Kumar, "Smart contracts and nfts: non-fungible tokens as a core component of blockchain to be used as collectibles," *Lecture Notes on Data Engineering and Communications Technologies. Springer Science and Business Media Deutschland GmbH*, vol. 73, pp. 401–422, 2022. 4.6

[28] W. Entriken, D. Shirley, J. Evans, and N. Sachs, "ERC-721: Non-Fungible Token Standard," *Ethereum Improvement Proposals*, 2018. [Online]. Available: https://eips.ethereum.org/EIPS/eip-721 4.6.1, 12.3

[29] E. G. Weyl, P. Ohlhaver, and V. Buterin, "Decentralized society: Finding web3's soul," *SSRN*, 2022. [Online]. Available: https://ssrn.com/abstract=4105763 4.6.2

[30] O. Aflak, P.-M. Le Bris, and M. Martin, "Erc-4671: Non-tradable tokens standard," *Ethereum Improvement Proposals*, 2022. [Online]. Available: https://eips.ethereum.org/EIPS/eip-4671 4.6.2

[31] M. Zoltu, "ERC-5114: Soulbound Badge," *Ethereum Improvement Proposals*, 2022. [Online]. Available: https://eips.ethereum.org/EIPS/eip-51142 4.6.2

[32] B. Cai, "ERC-5484: Consensual Soulbound Tokens," *Ethereum Improvement Proposals*, 2022. [Online]. Available: https://eips.ethereum.org/EIPS/eip-5484 4.6.2

[33] P. Servillo, "Soulbound nft (eip-4671)," *Medium*, 2022. [Online]. Available: https://medium.com/coinmonks/soulbound-nft-eip-4671-57674e705088 4.6.2

[34] C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, "Elliptic curve lightweight cryptography: A survey," *IEEE Access*, vol. 6, pp. 72 514–72 550, 2018. 4.7.3

[35] E. B. Barker, W. C. Barker, W. E. Burr, W. T. Polk, and M. E. Smid, "Recommendation for key management - part 1: General (revision 3)," *NIST special publication*, vol. 800, no. 57, pp. 1–147, 2012. 4.7.3, 14.4

[36] S. Chandel, W. Cao, Z. Sun, J. Yang, B. Zhang, and T.-Y. Ni, "A multi-dimensional adversary analysis of rsa and ecc in blockchain encryption," *Lecture Notes in Networks and Systems. Springer*, vol. 70, pp. 988–1003, 2020. 4.7.3

[37] L. Chen, D. Moody, A. Regenscheid, A. Robinson, and K. Randall, "Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters," *National Institute of Standards and Technology Special Publication SP 800-186*, 2019. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-186-draft 4.7.3

[38] International Organization for Standardization and International Electrotechnical Commission, "ISO/IEC 18033-2,Information Technology - Security Techniques - Encryption Algorithms - Part 2: Asymmetric Ciphers," 2006. 4.7.3, 1

[39] V. Gayoso Martínez, F. Hernández Álvarez, L. Hernández Encinas, and C. Sánchez Ávila, "A comparison of the standardized versions of ecies," *6th International Conference on Information Assurance and Security, IAS 2010*, pp. 1–4, 2010. 4.7.3, 11.2

[40] M. S. Christo, V. E. Jesi, U. Priyadarsini, V. Anbarasu, H. Venugopal, and M. Karuppiah, "Ensuring improved security in medical data using ECC and blockchain technology with Edge devices," *Security and Communication Networks*, 2021. 4.7.3

[41] B. Le Nguyen, E. L. Lydia, M. Elhoseny, I. V. Pustokhina, D. A. Pustokhin, M. M. Selim, G. N. Nguyen, and K. Shankar, "Privacy Preserving Blockchain Technique to Achieve Secure and Reliable Sharing of IoT Data," *Computers, Materials and Continua*, vol. 65, no. 1, pp. 87–107, 2020. 4.7.3

[42] T. Sampradeepraj, V. Anusuya Devi, and S. P. Raja, "Secure multicasting in wireless sensor networks using identity based cryptography," *Concurrency and Computation: Practice and Experience*, vol. 35, 2023. 4.7.4

[43] V. Kumar, S. Ray, D. Sadhukhan, J. Karmakar, and M. Dasgupta, "Enhanced pairing-free identity-based broadcast authentication protocol in wsn using elgamal ecc," *Security and Privacy*, vol. 6, no. 3, 2023. 4.7.4

[44] X. Yang, X. Chen, J. Huang, H. Li, and Q. Huang, "FS-IBEKS: Forward secure identity-based encryption with keyword search from lattice," *Computer Standards & Interfaces*, vol. 86, p. 103732, 2023. 4.7.4

[45] C. Rackoff and D. R. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack," *Advances in Cryptology — CRYPTO '91*, pp. 433–444, 1992. 4.7.5

[46] F. Hao, "Schnorr Non-interactive Zero-Knowledge Proof," *RFC 8235, September 2017*, 2017. [Online]. Available: https://tools.ietf.org/html/rfc8235 4.7.5, 4.7.5, 11.2, 11.2, 1

[47] X. Sun, F. R. Yu, P. Zhang, Z. Sun, W. Xie, and X. Peng, "A survey on zero-knowledge proof in blockchain," *IEEE Network*, vol. 35, no. 4, pp. 198–205, 2021. 4.7.5

[48] J. Hasan, "Overview and applications of zero knowledge proof (zkp)," *IJCSN-International Journal of Computer Science and Network*, vol. 8, no. 5, pp. 436–440, 2019. 4.7.5

[49] G. Munilla Garrido, M. Babel, and J. Sedlmeir, "Towards verifiable differentially-private polling," *Proceedings of the 17th International Conference on Availability, Reliability and Security (ARES '22). Association for Computing Machinery, New York, NY, USA. Article 6*, pp. 1–11, 2022. 4.7.5, 5.4

[50] A. M. Pinto, "An introduction to the use of zk-SNARKs in blockchains," *Mathematical Research for Blockchain Economy*, pp. 233–249, 2020. 4.7.5

[51] A. Banerjee, M. Clear, and H. Tewari, "Demystifying the role of zk-SNARKs in Zcash," *2020 IEEE Conference on Application, Information and Network Security (AINS)*, pp. 12–19, 2020. 4.7.5

[52] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," *2014 IEEE Symposium on Security and Privacy*, pp. 459–474, 2014. 4.7.5

[53] M. Bellés-Muñoz, M. Isabel, J. L. Muñoz-Tapia, A. Rubio, and J. Baylina, "Circom: A circuit description language for building zero-knowledge applications," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–18, 2022. 4.7.5

[54] M. Bellés-Muñoz, J. Baylina, V. Daza, and J. L. Muñoz-Tapia, "New privacy practices for blockchain software," *IEEE Software*, vol. 39, no. 3, pp. 43–49, 2022. 4.7.5

[55] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for zero-knowledge proof systems," *30th USENIX Security Symposium*, pp. 519–535, 2021. 4.7.5, 16.1, 2

[56] P. Murray, N. Welch, and J. Messerman, "EIP-1167: Minimal Proxy Contract," *Ethereum Improvement Proposals*, 2018. [Online]. Available: https://eips.ethereum.org/EIPS/eip-1167 4.8.2

[57] E. Politou, E. Alepis, C. Patsakis, F. Casino, and M. Alazab, "Delegated content erasure in IPFS," *Future Generation Computer Systems*, vol. 112, pp. 956–964, 2020. 4.9.1

[58] B. Guidi, A. Michienzi, and L. Ricci, "Data persistence in decentralized social applications: The IPFS approach," *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1–4, 2021. 4.9.1

[59] M. M. Madine, A. A. Battah, I. Yaqoob, K. Salah, R. Jayaraman, Y. Al-Hammadi, S. Pesic, and S. Ellahham, "Blockchain for giving patients control over their medical records," *IEEE Access*, vol. 8, pp. 193 102–193 115, 2020. 5

[60] A. A. Omar, M. Z. A. Bhuiyan, A. Basu, S. Kiyomoto, and M. S. Rahman, "Privacy-friendly platform for healthcare data in cloud based on blockchain environment," *Future Generation Computer Systems*, vol. 95, pp. 511–521, 2019. 5

[61] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Blockchain for secure ehrs sharing of mobile cloud based e-health systems," *IEEE Access*, vol. 7, pp. 66 792–66 806, 2019. 5

[62] M. Sookhak, M. Jabbarpour, N. Safa, and F. Yu, "Blockchain and smart contract for access control in healthcare: A survey, issues and challenges, and open issues," *Journal of Network and Computer Applications*, vol. 178, p. 102950, 2020. 5

[63] A. Z. Ourad, B. Belgacem, and K. Salah, "Using blockchain for iot access control and authentication management," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag*, vol. 10972, pp. 150–164, 2018. 5

[64] R. A. Memon, J. P. Li, J. Ahmed, M. I. Nazeer, M. Ismail, and K. Ali, "Cloud-based vs. blockchain-based iot: a comparative survey and way forward," *Frontiers of Information Technology and Electronic Engineering. Zhejiang University*, vol. 21, pp. 563–586, 2020. 5

[65] T. A. Butt, R. Iqbal, K. Salah, M. Aloqaily, and Y. Jararweh, "Privacy management in social internet of vehicles: Review, challenges and blockchain based solutions," *IEEE Access*, vol. 7, pp. 79 694–79 713, 2019. 5

[66] H. Hasan, E. AlHadhrami, A. AlDhaheri, K. Salah, and R. Jayaraman, "Smart contract-based approach for efficient shipment management," *Computers & Industrial Engineering*, vol. 136, pp. 149–159, 2019. 5

[67] Q. E. Abbas and J. Sung-Bong, "A survey of blockchain and its applications," *1st International Conference on Artificial Intelligence in Information and Communication, ICAIIC 2019. Institute of Electrical and Electronics Engineers Inc.*, pp. 1–3, 2019. 5

[68] R. W. Ahmad, K. Salah, R. Jayaraman, H. R. Hasan, I. Yaqoob, and M. Omar, "The role of blockchain technology in aviation industry," *IEEE Aerospace and Electronic Systems Magazine*, vol. 36, no. 3, pp. 4–15, 2021. 5

[69] J. Bao, D. He, M. Luo, and K.-K. R. Choo, "A survey of blockchain applications in the energy sector," *IEEE Systems Journal*, vol. 15, no. 3, pp. 3370–3381, 2021. 5

[70] A. Fatrah, S. El Kafhali, K. Salah, and A. Haqiq, "Transparent blockchain-based voting system: Guide to massive deployments," *Advances in Intelligent Systems and Computing*, vol. 1261, pp. 237–246, 2020. 5

[71] D. Li, W. E. Wong, M. Chau, S. Pan, and L. S. Koh, "A survey ofnfc mobile payment: Challenges and solutions using blockchain and cryptocurrencies," *2020 7th International Conference on Dependable Systems and Their Applications (DSA), Xi'an, China*, pp. 69–77, 2020. 5

[72] A. Vives-Guasch, M. Payeras-Capella, M. Mut-Puigserver, and J. Castellà-Roca, "E-ticketing scheme for mobile devices with exculpability," *Data Privacy Management and Autonomous Spontaneous Security*, pp. 79–92, 2011. 5

[73] A. Vives-Guasch, M. M. Payeras-Capellà, M. Puigserver, J. Castellà-Roca, and J. Ferrer-Gomila, "A secure e-ticketing scheme for mobile devices with near field communication (nfc) that includes exculpability and reusability," *IEICE Transactions*, vol. 95-D, pp. 78–93, 2012. 5

[74] M. Payeras-Capellà, M. Mut-Puigserver, J. Castellà-Roca, and J. Bondia-Barcelo, "Design and performance evaluation of two approaches to obtain anonymity in transferable electronic ticketing schemes," *Mobile Networks and Applications*, vol. 22, pp. 1137–1156, 2017. 5

[75] A. Vives-Guasch, M. M. Payeras-Capellà, M. Mut-Puigserver, J. Castellà-Roca, and J.-L. Ferrer-Gomila, "Anonymous and transferable electronic ticketing scheme,"

Data Privacy Management and Autonomous Spontaneous Security, pp. 100–113, 2014. 5

[76] W. Rehman, H. e. Zainab, J. Imran, and N. Z. Bawany, "Nfts: Applications and challenges," *2021 22nd International Arab Conference on Information Technology (ACIT)*, pp. 1–7, 2021. 5, 7.4

[77] M. M. Payeras-Capellà, M. Mut-Puigserver, J.-L. Ferrer-Gomila, J. Castellà-Roca, and A. Vives-Guasch, "Electronic ticketing: Requirements and proposals related to transport," *Advanced Research in Data Privacy*, vol. 567, pp. 285–301, 2015. 5

[78] M. Mut-Puigserver, M. M. Payeras-Capellà, J. L. Ferrer-Gomila, A. Vives-Guasch, and J. Castellà-Roca, "A survey of electronic ticketing applied to transport," *Computers & Security*, vol. 31, no. 8, pp. 925–939, 2012. 5

[79] G. Keyes, "Defi tops $100 billion for first time as cryptocurrencies surge," *Bloomberg*, 2021. [Online]. Available: https://www.bloomberg.com/news/articles/2021-10-20/defi-tops-100-billion-for-first-time-as-cryptocurrencies-surge 5

[80] The European Parliament and the Council of the European Union, "Regulation (eu) no 910/2014 of 23 july 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing," 2014, Directive 1999/93/EC. 5.1, 5.1.1, 5.2, 9.1

[81] Q. Huang, G. Yang, D. S. Wong, and W. Susilo, "A new efficient optimistic fair exchange protocol without random oracles," *International Journal of Information Security*, vol. 11, no. 1, pp. 53–63, 2012. 5.1, 8.1

[82] Q. Huang, D. S. Wong, and W. Susilo, "P2OFE: Privacy-preserving optimistic fair exchange of digital signatures," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag*, vol. 8366, pp. 367–384, 2014. 5.1, 5.2, 8.1

[83] Z. Shao, "Fair exchange protocol of schnorr signatures with semi-trusted adjudicator," *Computers & Electrical Engineering*, vol. 36, no. 6, pp. 1035–1045, 2010. 5.1, 8.1

[84] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," *Advances in Cryptology. CRYPTO 2014. Lecture Notes in Computer Science*, vol. 8617, pp. 421–439, 2014. 5.1, 8.1

[85] S. Goldfeder, J. Bonneau, R. Gennaro, and A. Narayanan, "Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag*, vol. 10322, pp. 321–339, 2017. 5.1, 8.1

[86] A. Küpçü and A. Lysyanskaya, "Usable optimistic fair exchange," *Topics in Cryptology - CT-RSA 2010. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg*, vol. 5985, pp. 252–267, 2010. 5.1, 8.1

[87] N. Asokan, V. Shoup, and M. Waidner, "Asynchronous protocols for optimistic fair exchange," *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 86–99, 1998. 5.1, 2, 2, 2

[88] J. A. Onieva, J. Zhou, and J. Lopez, "Multi-party non-repudiation: A survey," *ACM Computer Surveys*, vol. 41, p. 5, 2008. 5.1

[89] J. Zhou, R. Deng, and F. Bao, "Some remarks on a fair exchange protocol," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag.*, vol. 1751, pp. 46–57, 2000. 5.1

[90] J. Ferrer-Gomila, J. Onieva, M. M. Payeras-Capellà, and J. Lopez, "Certified electronic mail: Properties revisited," *Computers & Security*, vol. 29, no. 2, pp. 167–179, 2010. 5.1

[91] J. L. Ferrer-Gomila, M. Payeras-Capellà, and L. Huguet i Rotger, "An efficient protocol for certified electronic mail," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag.*, vol. 1975, pp. 237–248, 2000. 5.1, 8.2.2, 8.2.2, 8.3.2, 9.2.2

[92] S. Kremer and O. Markowitch, "Selective receipt in certified e-mail," *Progress in Cryptology. INDOCRYPT 2001. Lecture Notes in Computer Science*, vol. 2247, pp. 136–148, 2001. 5.1, 7.1

[93] M. Mut-Puigserver, J. L. Ferrer-Gomila, and L. Huguet-Rotger, "Certified e-mail protocol with verifiable third party," *Proceedings - 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service, EEE-05*, pp. 548–551, 2005. 5.1

[94] M. M. Payeras-Capellà, M. Mut-Puigserver, J. L. Ferrer-Gomila, and L. Huguet-Rotger, "No author based selective receipt in an efficient certified e-mail protocol," *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2009*, pp. 387–392, 2009. 5.1, 7.1

[95] J. L. Ferrer-Gomila, M. Payeras-Capellà, and L. Huguet-Rotger, "A realistic protocol for multi-party certified electronic mail," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag*, vol. 2433, pp. 210–219, 2002. 5.1, 9.2.2

[96] J. A. Onieva, J. Zhou, and J. Lopez, "Enhancing certified email service for timeliness and multicast," *Fourth International Network Conference*, pp. 327–335, 2004. 5.1

[97] S. Kremer and O. Markowitch, "A multi-party non-repudiation protocol," *Information Security for Global Information Infrastructures. SEC 2000. IFIP — The International Federation for Information Processing*, vol. 47, pp. 271–280, 2000. 5.1

[98] J. Zhou, J. Onieva, and J. Lopez, "Optimized multi-party certified email protocols," *Information Management and Computer Security*, vol. 13, pp. 350–366, 2005. 5.1

[99] S. Barber, X. Boyen, E. Shi, and E. Uzun, "Bitter to better - how to make bitcoin a better currency," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7397, pp. 399–414, 2012. 5.1, 8.1

[100] E. Heilman, F. Baldimtsi, L. Alshenibr, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted tumbler for bitcoin-compatible anonymous payments," *Network and Distributed System Security Symposium (NDSS)*, 2016. 5.1, 8.1

[101] E. Heilman, F. Baldimtsi, and S. Goldberg, "Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag*, vol. 9604, pp. 43–60, 2016. 5.1, 8.1

[102] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub," *Internet Society*, 2017. 5.1

[103] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Toward fairness of cryptocurrency payments," *IEEE Security & Privacy*, vol. 16, no. 3, pp. 81–89, 2018. 5.1, 8.6

[104] M. Al-Bassam, A. Sonnino, M. Król, and I. Psaras, "Airtnt: Fair exchange payment for outsourced secure enclave computations," *CoRR*, vol. abs/1805.06411, 2018. 5.1, 8.1

[105] S. Delgado-Segura, C. Pérez-Solà, G. Navarro-Arribas, and J. Herrera-Joancomartí, "A fair protocol for data trading based on bitcoin transactions," *Future Generation Computer Systems*, vol. 107, pp. 832–840, 2020. 5.1, 8.1

[106] H. R. Hasan and K. Salah, "Blockchain-based solution for proof of delivery of physical assets," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag*, vol. 10974, pp. 139–152, 2018. 5.1

[107] ——, "Proof of delivery of digital assets using blockchain and smart contracts," *IEEE Access*, vol. 6, pp. 65 439–65 448, 2018. 5.1

[108] C. Esposito, F. Palmieri, and K.-K. R. Choo, "Cloud message queueing and notification: Challenges and opportunities," *IEEE Cloud Computing*, vol. 5, no. 2, pp. 11–16, 2018. 5.1

[109] N. Zupan, K. Zhang, and H.-A. Jacobsen, "Hyperpubsub: A decentralized, permissioned, publish/subscribe service using blockchains: Demo," *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos (Middleware '17)*, pp. 15–16, 2017. 5.1

[110] H. K. Alper and A. Küpçü, "Optimally efficient multi-party fair exchange and fair secure multi-party computation," *ACM Transactions on Privacy and Security*, vol. 25, no. 1, pp. 1–34, 2021. 5.1

[111] European Union Agency for Cybersecurity, "Security guidelines on the appropriate use of qualified electronic registered delivery services," 2017. [Online]. Available: https://europa.eu/!G3JnTN 5.1.1

[112] European Council, "Directive 2000/31/ec on electronic commerce," 2000. 5.2, 13.2, 14.2

[113] N. Asokan, B. Baum-Waidner, M. Schunter, and M. Waidner, "Optimistic synchronous multi-party contract signing," *Research Report RZ 3089, IBM Research Division*, 1998. 5.2

[114] M. Payeras-Capellà, J. L. Ferrer-Gomila, and L. Huguet-Rotger, "Achieving fairness and timeliness in a previous electronic contract signing protocol," *Proceedings - First International Conference on Availability, Reliability and Security, ARES 2006*, pp. 717–722, 2006. 5.2, 13.1

[115] J. L. Ferrer-Gomila, M. Payeras-Capellà, and L. Huguet-Rotger, "Optimality in asynchronous contract signing protocols," *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3184, pp. 200–208, 2004. 5.2

[116] Z. Wan, R. H. Deng, and D. Lee, "Electronic contract signing without using trusted third party," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag*, vol. 9408, pp. 386–394, 2015. 5.2

[117] H. Tian, J. He, and L. Fu, "Contract coin: Toward practical contract signing on blockchain," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag*, vol. 10701, pp. 43–61, 2017. 5.2

[118] H. Huang, K.-C. Li, and X. Chen, "Blockchain-based fair three-party contract signing protocol for fog computing," *Concurrency and Computation: Practice and Experience. John Wiley and Sons Ltd*, vol. 31, 2018. 5.2

[119] O. Konashevych and O. Khovayko, "Online digital signing of deeds and contracts using the blockchain," *Ledger Journal*, 2018. 5.2

[120] L.-J. Guo and J. Li, Xuelianand Gao, "Multi-party fair exchange protocol with smart contract on bitcoin," *International Journal of Network Security*, vol. 21, pp. 71–82, 2019. 5.2

[121] A. P. Isern-Deyà, M. M. Payeras-Capellà, M. Mut-Puigserver, and J. L. Ferrer-Gomila, "Anonymous and fair micropayment scheme with protection against coupon theft," *International Journal of Adaptive, Resilient and Autonomic Systems*, vol. 4, no. 2, pp. 54–71, 2013. 5.3, 15, 15.4

[122] R. L. Rivest and A. Shamir, "Payword and micromint: Two simple micropayment schemes," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag*, vol. 1189, pp. 69–87, 1997. 5.3

[123] H. S. Galal, M. Elsheikh, and A. Youssef, "An efficient micropayment channel on ethereum," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer*, vol. 11737, pp. 211–218, 2019. 5.3

[124] M. Di Ferrante, "Ethereum payment channel in 50 lines of code," *Medium*, 2017. [Online]. Available: https://medium.com/@matthewdif/ethereum-payment-channel-in-50-lines-of-code-a94fad2704bc 5.3

[125] X. Luo, W. Cai, Z. Wang, X. Li, and C. M. Victor Leung, "A payment channel based hybrid decentralized ethereum token exchange," *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). Institute of Electrical and Electronics Engineers Inc.*, pp. 48–49, 2019. 5.3

[126] S. Tripathy and S. K. Mohanty, "Mappcn: Multi-hop anonymous and privacy-preserving payment channel network," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer*, vol. 12063, pp. 481–495, 2020. 5.3

[127] Raiden, "Raiden network," 2017. [Online]. Available: https://raiden.network/ 5.3

[128] K. Ryu, W. Kim, and E.-K. Lee, "Paygo: Incentive-comparable payment routing based on contract theory," *IEEE Access*, vol. 8, pp. 70 095–70 110, 2020. 5.3

[129] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments. draft version 0.5, 9, 14," 2016. [Online]. Available: https://lightning.network/lightning-network-paper.pdf 5.3, 15.1

[130] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," *International Conference on Financial Cryptography and Data Security. Springer*, pp. 201–226, 2020. 5.3, 15.4

[131] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," *Internet Society*, 2019. 5.3

[132] D. S. Baars, "Towards self-sovereign identity using blockchain technology," *University of Twente*, 2016. [Online]. Available: http://essay.utwente.nl/71274 5.4

[133] M. S. Ferdous, F. Chowdhury, and M. O. Alassafi, "In search of self-sovereign identity leveraging blockchain technology," *IEEE Access*, vol. 7, pp. 103 059–103 079, 2019. 5.4

[134] A. J. Zwitter, O. J. Gstrein, and E. Yap, "Digital identity and the blockchain: Universal identity management and the concept of the "self-sovereign" individual," *Frontiers in Blockchain*, vol. 3, 2020. 5.4

[135] A. Preukschat and D. Reed, *Self-Sovereign Identity: Decentralized digital identity and verifiable credentials*. Manning, 2021. 5.4

[136] P. J. Windley, *Learning Digital Identity: Design, Deploy, and Manage Identity Architectures*. O'Reilly Media, 2023. 5.4

[137] A. T. Sheik, C. Maple, G. Epiphaniou, and U. Atmaca, "A comparative study of cyber threats on evolving digital identity systems," *Competitive Advantage in the Digital Economy (CADE 2021)*, pp. 62–69, 2021. 5.4

[138] C. Allen, "The path to self-sovereign identity," 2016. [Online]. Available: http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html 5.4

[139] C. Mazzocca, A. Acar, S. Uluagac, R. Montanari, P. Bellavista, and M. Conti, "A survey on decentralized identifiers and verifiable credentials," 2024. 5.4

[140] A. Satybaldy, M. Nowostawski, and J. Ellingsen, "Self-sovereign identity systems: Evaluation framework," *Privacy and Identity Management. Data for Better Living: AI and Privacy. Privacy and Identity 2019. IFIP Advances in Information and Communication Technology. Springer, Cham*, vol. 576, pp. 447–461, 2020. 5.4

[141] V. Keršič, A. Vrečko, U. Vidovič, M. Domajnko, and M. Turkanović, "Using self-sovereign-identity principles to prove your worth in decentralized autonomous organizations," *Proceedings of the Ninth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications*, vol. 3237, 2022. [Online]. Available: https://ceur-ws.org/Vol-3237/paper-ker.pdf 5.4

[142] R. Mecozzi, G. Perrone, D. Anelli, N. Saitto, E. Paggi, and D. Mancini, "Blockchain-related identity and access management challenges: (de)centralized digital identities regulation," *2022 IEEE International Conference on Blockchain (Blockchain)*, pp. 443–448, 2022. 5.4

[143] T. J. Chaffer and J. Goldston, "On the existential basis of self-sovereign identity and soulbound tokens: An examination of the "self" in the age of web3," *Journal of Strategic Innovation and Sustainability*, vol. 17, no. 3, 2022. 5.4

[144] M. Eltuhami, M. Abdullah, and B. A. Talip, "Identity verification and document traceability in digital identity systems using non-transferable non-fungible tokens," *2022 International Visualization, Informatics and Technology Conference (IVIT)*, pp. 136–142, 2022. 5.4

[145] B. Yousra, S. Yassine, M. Yassine, S. Said, T. Lo'ai, and K. Salah, "A novel secure and privacy-preserving model for OpenID connect based on blockchain," *IEEE Access*, vol. 11, pp. 67 660–67 678, 2023. 5.4

[146] F. Hildebrandt, "The future of soulbound tokens and their blockchain accounts," no. 2, pp. 18–24, 2022. 5.4

[147] J. Goldston, T. J. Chaffer, J. Osowska, and C. von Goins II, "Digital inheritance in web3: A case study of soulbound tokens and the social recovery pallet within the polkadot and kusama ecosystems," 2023. 5.4

[148] U. Tejashwin, S. J. Kennith, R. Manivel, K. C. Shruthi, and M. Nirmala, "Decentralized society: Student's soul using soulbound tokens," *2023 International Conference for Advancement in Technology (ICONAT)*, pp. 1–4, 2023. 5.4

[149] K. Singh, O. Dib, C. Huyart, and K. Toumi, "A novel credential protocol for protecting personal attributes in blockchain," *Computers & Electrical Engineering*, vol. 83, p. 106586, 2020. 5.4

[150] N. Sun, Y. Zhang, and Y. Liu, "A privacy-preserving kyc-compliant identity scheme for accounts on all public blockchains," *Sustainability*, vol. 14, no. 21, 2022. 5.4

[151] R. Q. Saramago, H. Meling, and L. N. Jehl, "A Privacy-Preserving and Transparent Certification System for Digital Credentials," *26th International Conference on Principles of Distributed Systems (OPODIS 2022), Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 253, pp. 9:1–9:24, 2023. 5.4

[152] T. Wang, S. Zhang, and S. C. Liew, "Linking souls to humans with zkbid: Accountable anonymous blockchain accounts for web 3.0 decentralized identity," 2023. 5.4

[153] F. Baldimtsi, K. K. Chalkias, Y. Ji, J. Lindstrøm, D. Maram, B. Riva, A. Roy, M. Sedaghat, and J. Wang, "zklogin: Privacy-preserving blockchain authentication with existing credentials," 2024. 5.4

[154] A. A. Agarkar, M. Karyakarte, G. Chavhan, M. Patil, R. Talware, and L. Kulkarni, "Blockchain aware decentralized identity management and access control system," *Measurement: Sensors*, vol. 31, p. 101032, 2024. 5.4

[155] S. L. Nita and M. I. Mihailescu, "A novel authentication scheme based on verifiable credentials using digital identity in the context of web 3.0," *Electronics*, vol. 13, no. 6, 2024. 5.4

[156] Z. Song, E. Yan, J. Song, R. Jiang, Y. Yu, and T. Chen, "A Blockchain-Based Digital Identity System with Privacy, Controllability, and Auditability," *Arab J Sci Eng*, 2024. 5.4

[157] L. Zhou, A. Diro, A. Saini, S. Kaisar, and P. C. Hiep, "Leveraging zero knowledge proofs for blockchain-based identity sharing: A survey of advancements, challenges and opportunities," *Journal of Information Security and Applications*, vol. 80, p. 103678, 2024. 5.4

[158] Sovrin Foundation, "Sovrin: A protocol and token for self- sovereign identity and decentralized trust," *Sovrin*, 2018. [Online]. Available: https://sovrin.org/wp-content/uploads/Sovrin-Protocol-and-Token-White-Paper.pdf 5.4

[159] C. Lundkvist, R. Heck, J. Torstensson, Z. Mitton, and M. Sena, "Uport: A platform for self-sovereign identity," 2016. [Online]. Available: https://blockchainlab.com/pdf/uPort_whitepaper_DRAFT20161020.pdf 5.4

[160] N. N. Khalsa, C. Tuttle, K. Kahar, and S. El Damaty, "Holonym: Private proofs on identity for blockchains and beyond," 2022. [Online]. Available: https://docsend.com/view/6tgpdb375kwfgbzu 5.4

[161] "Iden3." [Online]. Available: https://iden3.io/ 5.4

[162] J. Diaz Rivera, A. Muhammad, and W.-C. Song, "Securing digital identity in the zero trust architecture: A blockchain approach to privacy-focused multi-factor authentication," *IEEE Open Journal of the Communications Society*, vol. 5, pp. 2792–2814, 2024. 5.4

[163] N. Prusty, *Building Blockchain Projects: Building decentralized Blockchain applications with Ethereum and Solidity*. Packt Publishing, 2017. 6

[164] N. Gonzalez-Deleito, "No author-based selective receipt in certified email with tight trust requirements," *Proceedings of 2005 International Workshop for Applied PKI, Singapore*, pp. 78–91, 2005. 7.1

[165] Q. Wang, R. Li, Q. Wang, and S. Chen, "Non-fungible token (nft): Overview, evaluation, opportunities and challenges," 2021. 7.4

[166] W. Radomski, A. Cooke, P. Castonguay, J. Therien, E. Binet, and R. Sandford, "ERC-1155: Multi Token Standard," *Ethereum Improvement Proposals*, 2018. [Online]. Available: https://eips.ethereum.org/EIPS/eip-1155 7.4

[167] H.-Y. Paik, X. Xu, H. M. N. D. Bandara, S. U. Lee, and S. K. Lo, "Analysis of data management in blockchain-based systems: From architecture to governance," *IEEE Access*, vol. 7, pp. 186 091–186 107, 2019. 7.4

[168] J. Liu, W. Li, G. O. Karame, and N. Asokan, "Towards fairness of cryptocurrency payments," 2016. 8.1, 2

[169] E. B. Barker and A. Roginsky, "Transitions: Recommendation for the transitioning of the use of cryptographic algorithms and key lengths," *Draft NIST Special Publication 800-131A*, 2019. 9.4

[170] N. I. of Standards and Technology, "Digital signature standard (dss)," *Federal Information Processing Standards Publication (FIPS) PUB 186-4*, 2013. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf 9.4, 14.4

[171] ——, "Secure hash standard (shs)," *Federal Information Processing Standards Publication (FIPS) PUB 180-4*, 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf 9.4

[172] S. for Efficient Cryptography Group (SECG), "Sec 2: Recommended elliptic curve domain parameters," *Standards for Efficient Cryptography (SEC)*, 2010. [Online]. Available: https://www.secg.org/sec2-v2.pdf 9.4

[173] I. O. for Standardization / International Electrotechnical Commission, "Iso/iec 18033-2:2006 information technology – security techniques – encryption algorithms – part 2: Asymmetric ciphers," 2006. 11.2

[174] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "Recommendation for key management part 1: General (rev. 3," *NIST special publication*, vol. 800(57), pp. 1–147, 2012. 11.4

[175] E. Barker, A. Roginsky, and R. Davis, "Recommendation for cryptographic key generation," *NIST Special Publication 800-133 Revision 2*, 2019. 2

[176] D. Anand, V. Khemchandani, and R. K. Sharma, "Identity-based cryptography techniques and applications (a review)," *2013 5th International Conference and Computational Intelligence and Communication Networks*, pp. 343–348, 2013. 3

[177] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," *SIAM Journal on Computing*, vol. 32, pp. 213–229, 08 2001. 12.2

[178] S. Duangphasuk, P. Duangphasuk, and C. Thammarat, "Fair exchange protocol in electronic transactions revisited," *2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pp. 331–334, 2020. 12.6

[179] T. Górski, "Integration flows modeling in the context of architectural views," *IEEE Access*, vol. 11, pp. 35 220–35 231, 2023. 12.6

[180] L. Chen, D. Moody, A. Regenscheid, A. Robinson, and K. Randall, "Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters," *Draft NIST Special Publication 800-186*, 2019. 14.3

[181] C. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, pp. 161–174, 1991. 1

[182] J. Baylina, "Ecsol. elliptic curve implementation in solidity." [Online]. Available: https://github.com/jbaylina 14.4

[183] V. Buterin, Y. Weiss, D. Tirosh, S. Nacson, A. Forshtat, K. Gazso, and T. Hess, "EIP-4337: Account Abstraction Using Alt Mempool," *Ethereum Improvement Proposals*, 2021. [Online]. Available: https://eips.ethereum.org/EIPS/eip-4337 2